PC Business INFO

DOS SPECIAL

Uitgebreide MS-DOS gebruikersgids

Nr. 3 1991

PC Business Info
Jaargang 6, nr. 5

ƒ 15,95 / Bfr 325

Overstap naar Clipper 5.0

TurboPascal voor Windows

MS-DOS versie 5.0

TurboVision



computercollectief

microcomputer tijdschriften boeken en software

in BELGIE is alles verkrijgbaar bij:
HET COMPUTERWINKELTJE MECHELEN
M Sabbestraat 39, B-2800 Mechelen
tel: 015-20 66 45 fax: 015-20 73 32
HET COMPUTERWINKELTJE BRUGGE
Moerkerksesteenweg 241, B-8310 Brugge
tel: 050-37 09 61 fax: 050-36 16 55

COMPUTERBOEKEN Top 30 Juni 1991

Basishandleiding WordPerfect 5.1 15
Werken met WordPerfect 5.1 (Boeke) .. 69
Leidraad DrawPerfect 1.1 12,95
*CLIPPER 5.0 - SYBEX Professioneel ... 99
Handleiding DOS 4.01 (Oets) 49
*Clipper Programming Guide, 2nd 5.0 .. 79
Basishandleiding Lotus 1-2-3 2.2 en 3 15
Basishandleiding DOS & Hard Disk 15
Leidraad DOS 4.0 12,95
WordPerfect 5.1 Gebruikersboek (Veen) 65

Werken met Windows 3.0 NL 69
Clipper 5.0 Developers Library + disk 99
Het Complete DOS 4.0 Boek 69
Basiscursus WordPerfect 5.1 NL 29,50
Werken met WordPerfect 5.1 NL (QUE) 99,50
Het Grote LARRY Boek 29,50
*Basishandleiding Windows 3 15
*Basishandleiding WordPerfect Gevorderd 15
*Hintbook King's Quest V 25
*ResEdit Complete - incl ResEdit 2.1 79

Tekenen met Dr.Halo/Dr.Genius 22,50
Gids voor Microsoft Works 2.0 NL .. 59,50
*Undocumented DOS 99
Leidraad PC Tools 6 12,95
Programming Windows 3 79
Werken met PC Tools Deluxe 6.0 69
Using Clipper, 2nd Ed - through 5.0 . 69
Grote MS-Flightsimulator 3/4 boek . 49,90
Basiscursus MS-DOS 29,50
Starten met MS-DOS/PC-DOS 35

Nieuw binnengekomen (*) en actuele boeken

DOS, OS/2, Windows
*QuickStart MS-DOS 5.0 29
*Werken met DR DOS 5.0 49
*Werken met Windows 3 64,50
*Het Kleine Windows 3 Boek .. 29,50
*Windows 3.0/SYBEX Professioneel 69
Power of PenPoint 59

Programmeertalen
TSR Programma's onder DOS .. 59,50
BASIC PowerTools 119
*Advanced BASIC - PDS and Quick. 99
QuickBASIC Toolbox 99
*Using Borland C++ 79
Encyclopedia C for MS-C 6.0 .. 79
*T-Pascal 6.0 Programmeergids 59,50
*Mastering Turbo Pascal 6 69

Amiga, Macintosh
*AmigaWorld AmigaDOS 2.0 Comp... 65
*Amiga Bridgeboard Buch 69
*Microsoft Excel 3 Mac Companion 79
*System 7 Book 65
*QuarkXPress Book 69
Macintosh Pascal Programming
Primer I - Using THINK Pascal 69

PC
*Fix your own PC 65
Progr Guide to EGA/VGA/SuperVGA 79
PC Reparatiegids 34
*PC-Hardware - assemblage ... 42,50

Databases, Spreadsheets
*dBASE IV Handboek - versie 1.1 85
*Werken met Clipper 5.0 deel 1 89
*Clipper Compleet 5.01 79,50
*Using Clarion Professional ... 69
*Running Excel 3.0 69
*Werken met R:BASE 3.1 69
*Mastering Quattro PRO 3 65

CAD
*AutoCAD 11 Handboek 2D tekenen 59
*Maximizing AutoCAD II: AutoLISP 89
*Generic CADD 5.0 Inside & Out 69

WordProcessors, DTP
*Using LetterPerfect 59
*WordPerfect 5.1 Cookbook +disk 69
*Microsoft WordBasic Primer ... 65
*Mastering PageMaker 4 IBM PC . 59
*Praktijkboek Ventura 3.0 49
*Using Ventura 3.0 GEM & Windows 79

Graphics, Utilities, diversen
*Hintbook Space Quest IV 25
*Leren vliegen Flightsimulator 4 38
*Official Book of King's Quest 39
*Inside Corel DRAW 2.0 69
*Werken met DrawPerfect 1.1 ... 49
*PC Magazine Guide Connectivity 99
Master SimCity/SimEarth 59
Using Carbon Copy PLUS 59
*Snelcursus Norton Utilities 5 29

N I E U W B I N N E N G E K O M E N (*) E N A C T U E L E S O F T W A R E (inclusief BTW)

Amiga
Disney AnimationStudio 369
*3D Construction Kit .. 179
*AmigaVision 349
Das Boot 99
Centurion 89
*ChessMaster 2100 99
*Chuck Yeager 2.0 89
*Eye of the Beholder . 99
*European Superleague . 89
*Gods 89
*Harpoon 99
LARRY I, II & III Pack 159
*MegaTraveller 1 99
*Quest for Glory II ... 129
*Secret of Monkey Island 89
*Switchblade II 89
*Wonderland 99

Mac
*Battle Chess 99
*Studio/1 299
*Studio/8 version 2.0 . 599
*Studio/32 1499

ST
HighSpeed Pascal 1.1 . 299
*3D Construction Kit .. 149
*Advanced Destroyer ... 89
*Elvira 99
*Lemmings 89
*Secret of Monkey Island 89
*Supercars II 89
*Wonderland 99

****
MS-DOS 5.0 International
UPDATE, 5.25" of 3.5" 199
****

PC SOFTWARE TOP 20
Flightassignment A.T.P.129
Aircraft&Scenery Design 99
Flightsimulator 4.0 .. 129
EZCosmos 3.0 149
Scenery Disk W. Europe 59
Space Quest IV 129
QEMM-386 5.12 219
Red Baron VGA (HD) ... 129
Copy II PC 6.0 99
SoundBlaster soundcard 489

Entertainm. Pk Windows 89
Norton Utilities 5.0 299
*Microsoft GameShop ... 99
*King's Quest V 129
AdLib Synthesizer Card 229
*SimEarth 139
Railroad Tycoon 119
DR DOS 5.0 389
ChessMaster 2100 89
Silent Service II 119

PC software voor Kinderen
Donald's Alphabet Chase 69
Goofy's Railway Express 69
Mickey's Runaway Zoo . 69
Mickey's 123's 99
Mickey's Colors&Shapes 99

PC toepassingen (incl BTW):
*3D Construction Kit .. 179
askSam 5.0 949
*Ami Pro voor WindowsNL 1299
Bannermania 89
*dGE 4.1 799
*Facelift for WordPerfect249
*GeoWorks Ensemble 489
*Play it by Ear 289
*Quattro PRO 3.0 799
Sidekick 2.0 199
*SuperCalc 5 469
VERTAAL! voor WP 349

PC utilities (incl BTW):
Adobe Type Manager ... 239
Adobe PLUS Pack 469
BeckerTools 2.0 Windows 289
Bitstream Facelift ... 249
*Central Point ANTIVIRUS 399
*Disklock PC 479
Entertainment Pack Win3 89
FastLynx 1.1 359
FileApps for Windows . 299
*Freedom of Press Light 239
HiJaak 2.0 449
*Laptop UltraVision ... 199
PC-Kwik PowerPak 2.0 . 299
*PC TOOLS DELUXE 7.0 .. 379
*UPDATE PC TOOLS 7.0 .. 153
*PrintQ 5.0 379
*Publisher's Paintbrush 1099
*ToolBook 1.5 f Windows 1099
Turbo EMS 6.0 259

PC talen (incl BTW):
GFA BASIC fuer MS-DOS . 599
*CodeBase++ 859
*Microsoft MASM 6.0 .., 399
*Borland C++ 799
*TurboPascal for Windows 499
Microsoft C 6.0 999
*SmallTalk/V Windows .. 1199
*Zortech C++ DatabaseLib 799

PC muziek, geluid (inc BTW):
Copyist DTP - score ed. 789
Sequencer Plus GOLD .. 599
*AdLib MCA card 429

PC games (incl BTW):
*Advanced Destroyer Sim. 89
*Bushbuck Charms, etc .. 139
*Champion of the Ray ... 89
*Death Knights of Krynn 99
*Elite Plus 139
*European SuperLeague .. 89
*F-29 Retaliator 119
*Heart of China VGA 139
*Jetfighter II 139
*Keys to Maramon 99
LARRY I,II & III Pack 159
*Lemmings 119
*Lexi-Cross 99
*Magic Candle 1 99
*MegaTraveller 1 119
MIG-29 Fulcrum 139
*Operation Stealth VGA 99
Sorcerers get all girls 99

Winkel open van dinsdag t/m zaterdag tussen 10 en 5 (maandag gesloten)
Alle prijzen zijn inclusief BTW - verzendkosten f 6,- per bestelling

Amstel 312 (t.o. Carré) 1017 AP Amsterdam Fax (020) 226668 Postbank 4475158 NMB 697915646

Vraag onze GRATIS ZOMERCATALOGUS aan! *
* We sturen hem GRATIS toe als je ons een *
* kaartje stuurt met je naam en adres. *
* Vermeld tevens: 'MS DOS SPECIAL' *

prijswijzigingen voorbehouden - LET OP: ons faxnummer is nu 622 6668

dealer aanvragen welkom

Redactioneel

Een nieuwe DOS Special met weer een gevarieerde inhoud. We hebben getracht in dit nummer de lijn aan te houden die we hebben uitgezet in de vorige DOS Special. Enkele artikelen over Windows-programmeren, object-georiënteerd programmeren en besprekingen van nieuwe produkten op het vakgebied.

Tevens een bespreking van de nieuwe Clipper compiler versie 5.0 in dit nummer. Enkele eigenschappen van deze compiler worden grondig behandeld, alsmede of de overstap de moeite waard is.

We streven ernaar om elk nummer aandacht te besteden aan de wat minder bekende, zij het exotische programmeertalen. De object-georiënteerde talen Actor en Eiffel worden in dit nummer behandeld; de volgende keer zullen we Oberon en Modula-3 onder de loep nemen. Actor is een taal die gebruikt kan worden om Windows-applicaties te schrijven. Eiffel is een taal die gebruikt kan worden om programma's te schrijven die een zeer hoge mate van software-betrouwbaarheid moeten hebben.

De redactie heeft vorige maand de Programmer's Conference van Borland in San Francisco bezocht. De indruk die we al hadden werd daar wederom bevestigd. Het Windows en object-georiënteerde programmeren zit in de lift, en nieuwe produkten zoals Turbo Pascal for Windows bieden nieuwe mogelijkheden op programmeergebied. Bij het spreken met de daar aanwezige programmeurs kregen we het idee dat men de Windows-golf die nu over ons heen komt met de nodige scepsis bekijkt, maar dat men er wel erg geïnteresseerd in is. Het wachten voor de C- en C++programmeurs is natuurlijk op Borland C++ for Windows, maar de introductie daarvan zal nog minstens tot oktober op zich laten wachten. Zodra er ook maar iets meer over deze compiler bekend is wordt dit uiteraard in de DOS Special gemeld. Het grote voordeel is natuurlijk dat deze versie Windows-hosted is, maar ook zal dan de ObjectWindows-library worden meegeleverd die het programmeren van Windows-applicaties een stuk eenvoudiger maakt. Deze is wel al voor Pascal-programmeurs beschikbaar in Turbo Pascal for Windows.

In de vorige DOS Special werd al een klein overzicht gegeven van de nieuwste DOS-versie. In dit nummer worden alle eigenschappen en nieuwe commando's uitgebreider besproken. Het lijkt erop alsof Microsoft het geheugenprobleem van eerdere DOS-versies erkent. Met 5.0 is het mogelijk om device-drivers en DOS zelf 'hoog' te laden. Dit betekent dat het gebied tussen 640 en 1024kB gebruikt wordt om het vrije conventionele geheugen te vergroten. Echt nieuw is het natuurlijk niet, DR DOS bevatte deze mogelijkheid ook al. Microsoft is erin geslaagd om tot maar liefst 630kB conventioneel geheugen vrij te maken.

De laatste processor die Intel op de markt heeft gebracht is de i486SX. Dit goedkopere type van de 'echte' 80486 ontbeert een ingebouwde coprocessor en draait op een lagere kloksnelheid. Daardoor is de chip een stuk goedkoper; een ander voordeel van de 486 is een ingebouwde cache. Enkele computerfabrikanten hebben al een model met de 486SX, waaronder het Hollandse bedrijf Tulip. Op pagina 104 vindt u een beschrijving van deze processor.

De redactie

Colofon

DOS Special 3/91

Uitgave:
Sala Communications
Postbus 43048
1009 ZA Amsterdam

Uitgever:
ing. V. Sala

Hoofdredacteur:
ir. L. Sala
Eindredactie:
John Caspers, Wilbert Scheer
Redactie:
Sjoerd van Gelderen
André van der Hoeven
tel: 020-6228871

Produktie:
Jeroen Broekhuizen, Patricia Buenting, drs. Hein Zoete

Aan dit nummer werkten mee: *Jeannot Bos, Joost Burgering, John Caspers, Wiebe van Erkelens, Sjoerd van Gelderen, Eva Hilhorst (fotografie en illustraties), André van der Hoeven, Barry Lagerweij, H. Leeflang, Mike Lewis, Dennis ten Siethoff en anderen.*

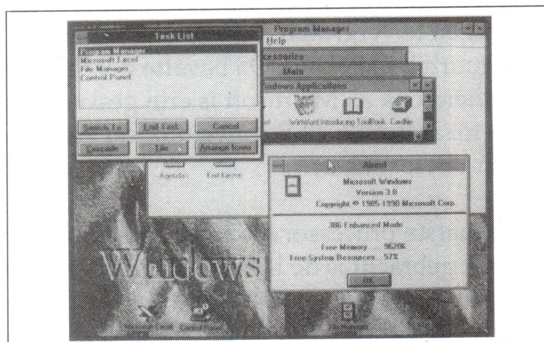
Abonnementen:
Marjo Jansen
tel.: 020 - 6248006

Advertenties:
V. Sala, H. Bia, B. Sala
tel: 020-6273198

Druk en zetwerk:
NDB, Zoeterwoude

Distributie:
In Nederland: Betapress
In België AMP, Brussel

Inhoud DOS Special nummer 3/91



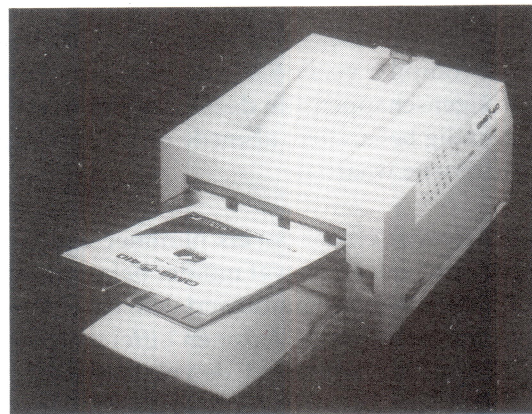
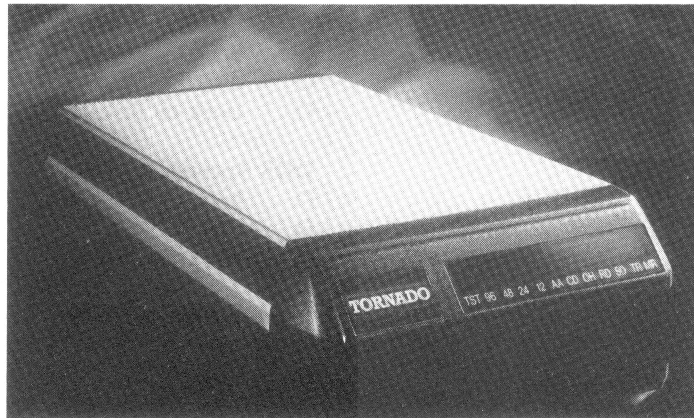




foto van pagina 104 (Intel)

unieke ***DOS Special*** MODEM *aanbieding*

Nederlands grootste modem-leverancier, G & B Computers BV, start in samenwerking met SALA Communications een modem-actie.
Ontdek met uw eigen PC en een Tornado de onbegrenste mogelijkheden van het telefonisch data-verkeer. Voor deze actie hebben wij een zorgvuldige keuze gemaakt uit het leveringsprogramma van Tornado; de modems zijn goedgekeurd door het Ministerie van Verkeer en Waterstaat en voorzien van een duidelijke handleiding.



✔ **TORNADO I**
Een intern modem met het V21, V22 en V22 bis protocol. Eenvoudig te installeren. En u kunt optimaal gebruik maken van onder andere de diensten van Videotex.
Adviesprijs *f* 349,-. **Actieprijs** *f* 299,–

✔ **TORNADO I EXTERN**
Een modem met dezelfde specificaties als de Tornado I alleen dan de externe uitvoering.
Adviesprijs *f* 399,-. **Actieprijs** *f* 349,–

✔ Top of the line, het nieuwste **TORNADO 9600 EV** externe modem
Met naast V21, V22 en V22 bis, ook nog V23, V32 en V42 bis. Tevens beschikt dit modem over de MNP-5 data-correctie mogelijkheid. In de praktijk betekent dit dat men met behulp van dit modem een transfer-rate van 38.400 BPS kan bereiken.
Adviesprijs *f*1.995,-. **Actieprijs** *f* 1.795,-

(Prijzen zijn inclusief btw.)

Modem-actie bestelbon

Naam : ____________________________

Adres : ____________________________

Postcode/Plaats : ______________________

wil het volgende modem ontvangen:

☐ **Tornado I** (*f* 299,-)
☐ **Tornado I extern** (*f* 349,-)
☐ **Tornado 9600 EV extern** (*f* 1.795,-)

en zal de betaling als volgt regelen:

☐ Ik maak het juiste bedrag *f* .... over op girorekening 5206360 t.n.v. Sala Communications, Amsterdam onder vermelding van MODEM-ACTIE.
☐ Ik verwacht een factuur en krijg dit modem na betaling toegestuurd.
☐ Ik stuur bijgaand een cheque voor het juiste bedrag.

Voorwaarden: Aanbieding geldt alleen voor abonnees. Geldig tot 1 september 1991

Dos Special Bestelbon voor Nabestellingen

Deze bon kunt u gebruiken om oude nummers en/of de bijbehoerende diskettes te bestellen.
Hieronder treft u een overzicht aan van de na te bestellen nummers. Kruist u s.v.p. aan wat u wilt bestellen. U kunt bij elk item zelf de prijs invullen en vervolgens alles optellen. Op deze manier weet u precies welk bedrag u dient over te maken.
De prijzen zijn als volgt.

° DOS Special **Boek** *f* **15,95** of **325** Bfr.
° DOS Special **Disk** *f* **14,50** of **300** Bfr.
° DOS Special Boek en diskette samen**:**
***f* 24,95** of **500** Bfr **.**
U kunt nu zelf uw bestelling en het totaal bedrag samenstellen.

Naam:

Telefoon:

Adres: .

Postcode/plaats:

TYPE diskette:
O 5,25 inch
O 3,5 inch *
(doorhalen wat niet gewenst wordt)

TOTAAL BEDRAG van mijn bestelling:

f

Stuur deze bon (of kopie) in een envelop met een onderteкende girobetaalkaart of bankcheque naar:
Sala Communications, Postbus 43048,
1009 ZA Amsterdam.
U kunt ook bestellen door direct het verschuldigde bedrag over te maken op giro 1585491 t.n.v. Sala Communications, Amsterdam. U hoeft dan deze bon ***NIET*** *op te sturen,* ***WEL*** *moet u de gewenste nummers en diskettes vermelden. (Bijv.: nr. 4/90 disk en boek.)*
Voor België: BBL 310-0506025-62

DOS Special nr. 3/89:
O boek *f*
O disk *f*
O Boek en diskette *f*

DOS Special nr. 4/89:
O boek *f*
O disk *f*
O Boek en diskette *f*

DOS Special nr. 1/90:
O boek *f*
O disk *f*
O Boek en diskette *f*

DOS Special nr. 2/90:
O boek *f*
O disk *f*
O Boek en diskette *f*

DOS Special nr. 3/90:
O boek *f*
O disk *f*
O Boek en diskette *f*

DOS Special nr. 4/90:
O boek *f*
O disk *f*
O Boek en diskette *f*

DOS Special nr. 1/91:
O boek *f*
O disk *f*
O Boek en diskette *f*

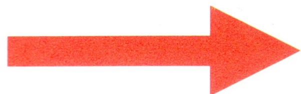
DOS Special nr. 2/91:
O boek *f*
O disk *f*
O Boek en diskette *f*

DOS Special nr. 3/91:
O boek *f*
O disk *f*
O Boek en diskette *f*

Totaal *f*

LET OP: DOS Special nr. 1 en 2/89 zijn niet meer leverbaar.

Neem nú een abonnement op de DOS Special en u weet u verzekerd van de volgende nummers van DOS Special



Tevens ontvangt u dan PC Business Info minstens zes maal per jaar er extra bij.

Abonnement DOS Special/PC Business Info

JA, ik wil een combinatie-abonnement DOS Special/PC Business Info. Ik krijg per jaar minimaal vier nummers DOS Special en zes nummers PC Business Info voor *f* 75,-. (Bfr. 1500)

Naam:.. Telefoon: ....

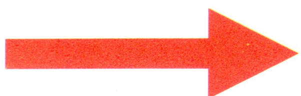
Adres:...

Postcode/plaats: ....

Stuur deze bon (of kopie) in een envelop met een ondertekende girobetaalkaart of bankcheque naar: **Sala Communications, Postbus 43048, 1009 ZA Amsterdam.**

U kunt ook bestellen door direct het verschuldigde bedrag over te maken, dan hoeft u deze bon NIET op te sturen, op giro 1585491 t.n.v. Sala Communications, Amsterdam o.v.v. het gewenste abonnement. Voor België geld overmaken op BBL 310-0506025-62.

Neem nú een abonnement op de DOS Special en u weet u verzekerd van de volgende nummers van DOS Special



Tevens ontvangt u dan de bijbehorende diskette met alle listings uit de DOS Special.

Abonnement DOS Special/Diskette

JA, ik wil een combinatie-abonnement DOS Special/Diskette. Ik krijg per jaar minimaal vier nummers DOS Special en de bijbehorende diskettes voor *f* 75,-. (Bfr. 1500)

Naam:.. Telefoon: ....

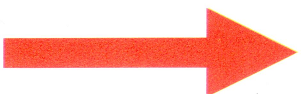
Adres:...

Postcode/plaats: ....

Stuur deze bon (of kopie) in een envelop met een ondertekende girobetaalkaart of bankcheque naar: **Sala Communications, Postbus 43048, 1009 ZA Amsterdam.**

U kunt ook bestellen door direct het verschuldigde bedrag over te maken, dan hoeft u deze bon NIET op te sturen, op giro 1585491 t.n.v. Sala Communications, Amsterdam o.v.v. het gewenste abonnement. Voor België geld overmaken op BBL 310-0506025-62.

Neem nú een abonnement op de DOS Special en u weet u verzekerd van de volgende nummers van DOS Special



Tevens ontvangt u dan PC Business Info minstens zes maal per jaar er extra bij en de DOS Special Diskettes.

Abonnement DOS Special/PC Business Info/Diskette

JA, ik wil een combinatie-abonnement DOS Special/PC Business Info/Diskette. Ik krijg per jaar minimaal vier nummers DOS Special en de bijbehorende diskettes én zes nummers PC Business Info voor *f* 115,-. (Bfr. 2300)

Naam:.. Telefoon: ....

Adres:...

Postcode/plaats: ....

Stuur deze bon (of kopie) in een envelop met een ondertekende girobetaalkaart of bankcheque naar: **Sala Communications, Postbus 43048, 1009 ZA Amsterdam.**

U kunt ook bestellen door direct het verschuldigde bedrag over te maken, dan hoeft u deze bon NIET op te sturen, op giro 1585491 t.n.v. Sala Communications, Amsterdam o.v.v. het gewenste abonnement. Voor België geld overmaken op BBL 310-0506025-62.

PC Starter

Informatief handboek voor de beginnende computergebruiker

DERDE
EDITIE
Prijs: *f* 14,95
ISBNnr. 90-240-0723 2

PC Starter is bedoeld voor iedereen die van start gaat met een MS-DOS Personal Computer. Het is samengesteld op basis van een bekende artikelen serie in het toonaangevende computermagazine Computer Info. In eenvoudige bewoordingen worden vaktermen en afkortingen uit de automatisering stap voor stap behandeld.



Een ideaal boek voor wie zich op een snelle manier alle benodigde basisbegrippen voor het PC gebruik eigen wil maken. Het boek is zeer geschikt voor gebruik in het onderwijs. Alle hoofdstukken worden vergezeld met vragenlijsten, die voor evaluatiedoeleinden kunnen worden gebruikt.

Onderwerpen: Hardware, Monitoren; Opslagmedia; Controls; Uitbreidingskaarten; Videokaarten; Printers; Datacommunicatie; Software; Besturingssysteem; Tekstverwerking en DTP, incl. WordPerfect; Databases. incl. dBase; Spreadsheets, incl. Lotus 1-2-3; PS/2-computers; Netwerken.

U kunt deze uitgave bestellen door overmaken van *f* 16,50 (incl verzendkosten)op gironummer 1585491 t.n.v. SAC Amsterdam onder vermelding van: PC Starter. Voor het onderwijs zijn er kortingstarieven. Leraren Informatica kunnen een recensie-exemplaar aanvragen via

Sala Com./ t.a.v. M. Jansen,Postbus 43048, 1009 ZA Amsterdam.

Object georiënteerd programmeren in Turbo Pascal

Deel 2: Virtuele methoden en dynamische objecten

In de vorige DOS-Special werd een introductie gegeven in object-georiënteerd programmeren (OOP) in Turbo Pascal. Er werd gesproken over encapsulation, inheritance en polymorfie. Over deze drie steunpijlers van OOP was toen natuurlijk nog lang niet alles gezegd, daar is de materie gewoon te uitgebreid voor. Virtuele methoden zijn in dat artikel in het geheel niet behandeld, terwijl ze toch een wezenlijk onderdeel van OOP vormen.

Wanneer een methode virteel is betekent dat informeel: gebruik de body van de methode in het afgeleide object, indien dat mogelijk is.

Om het geheugen nog even op te frissen, een object is een verzameling van datamembers en methoden die bewerkingen op die datamembers kunnen uitvoeren. Een object kan de methoden en datamembers van een ander object erven, men spreekt dan van inheritance. Er kunnen aan dat object nieuwe methoden en datamembers worden toegevoegd en de bestaande methoden uit het ouderobject mogen overschreven worden. Dat wil zeggen, de methode houdt dezelfde naam, maar de body kan verschillend zijn en de parameters mogen verschillen. Een variabele van dat type object, een zogenaamde instantie of instance, krijgt dan bij een aanroep naar een methode de voor hem bedoelde.

Maar in het volgende voorbeeld zal de uitvoer niet zijn zoals we zouden willen. De bedoeling is om de naam van beide objecten op het beeldscherm te krijgen, maar er wordt twee keer 'BaseObject' geprint.

```
Type  BaseObject
            = Object
               Name   : String;
               Procedure PrintMyName;
               Procedure Print;
              End;
      InheritedObject
            = Object (BaseObject)
               Procedure PrintMyName;
              End;

Procedure BaseObject.PrintMyName;
Begin
 Name:='BaseObject';
 WriteLn (Name);
End;

Procedure BaseObject.Print;
Begin
  PrintMyName;
End;

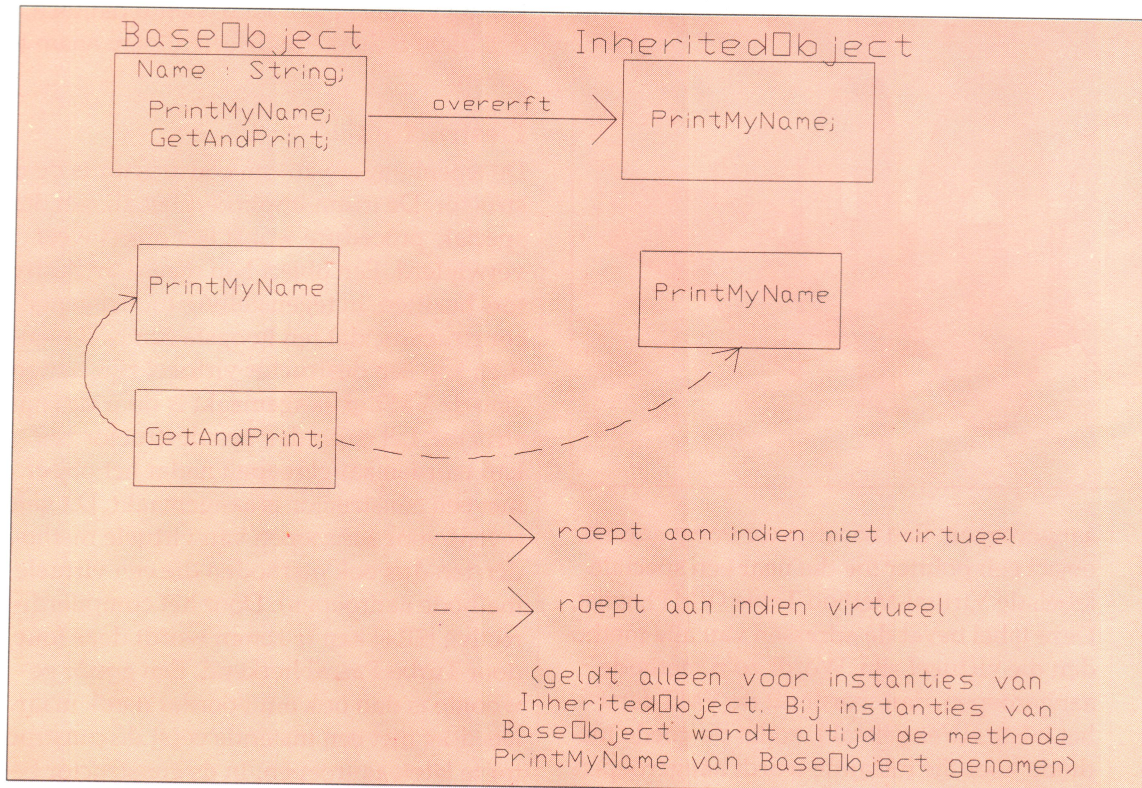
Procedure InheritedObject.PrintMyName;
Begin
 Name:='InheritedObject';
 WriteLn (Name);
End;

Var Base : BaseObject;
    Inherited : InheritedObject;

Begin
 Base.Print;
 Inherited.Print;
End.
```

De oorzaak hiervan kunt u terugvinden in de volgende tekening op de volgende pagina.

InheritedObject erft de methoden 'PrintMyName' en 'Print' van BaseObject. De methode 'PrintMyName' wordt echter overschreven. Het zou dus logisch zijn dat wanneer de instantie 'Inherited' de message 'Print' stuurt, de methode 'PrintMyName' van 'InheritedObject' zou worden genomen. Toch gebeurt dit niet, en het waarom is vrij eenvoudig in te zien. De compiler moet namelijk bij het vertalen van de methode 'Print' weten naar welk adres gesprongen moet worden wanneer de methode 'PrintMyName' wordt uitgevoerd. Hij neemt hiervoor de methode 'PrintMyName' die in 'BaseObject' is gedefi-



niëerd. Dit is volstrekt logisch, want voor een instantie van 'BaseObject' moet ook deze 'PrintMyName' worden aangeroepen. Deze manier van binding staat bekend onder 'early binding' of 'vroege binding'. Gelukkig bezit Pascal ook de mogelijkheid om het tegenovergestelde, 'late binding', te verwezenlijken. Late binding wil dus zeggen dat pas tijdens de executie van het programma wordt besloten welke versie van de methode moet worden aangeroepen. Deze methoden heten virtueel en worden als zodanig gekenmerkt indien achter hun declaratie in het object het keyword 'virtual' staat.

```
Type  BaseObject
        = Object
           Name   : String;
           Procedure PrintMyName;Virtual;
           Constructor Print;
          End;
      InheritedObject
        = Object (BaseObject)
           Procedure PrintMyName;Virtual;
          End;

Procedure BaseObject.PrintMyName;
Begin
 Name:='BaseObject';
 WriteLn (Name);
End;

Constructor BaseObject.Print;
Begin
 PrintMyName;
End;
```

```
Procedure InheritedObject.PrintMyName;
Begin
 Name:='InheritedObject';
 WriteLn (Name);
End;

Var Base : BaseObject;
    Inherited : InheritedObject;

Begin
 Base.Print;
 Inherited.Print;
End.
```

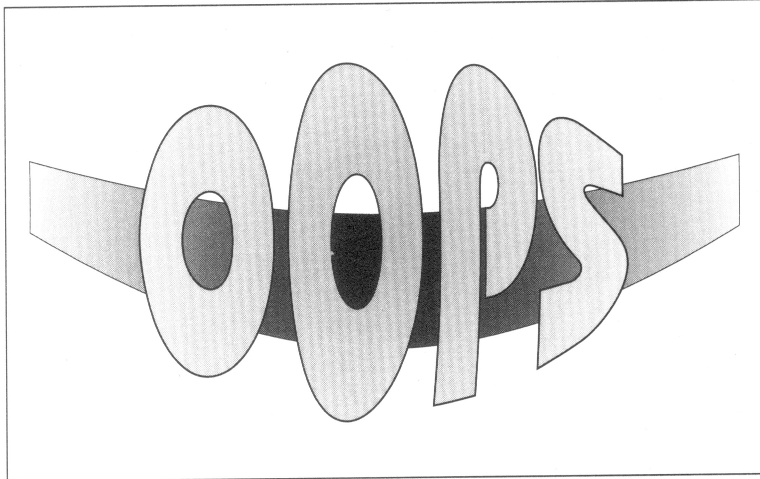
De uitvoer van deze code zal wel het beoogde resultaat geven, en wel:

```
BaseObject
InheritedObject
```

U ziet dus dat hier gebeurd wat in het begin van het artikel vermeld staat. De body van de methode van het afgeleide object wordt gebruikt als dat mogelijk is.

Constructor

Er is nu nog een nieuw keyword bijgekomen, namelijk 'constructor'. Een constructor is in feite niets ander dan een procedure, maar met een speciale eigenschap. Wanneer er virtuele methoden in een object zijn gedeclareerd moet alvorens met een instantie gewerkt wordt, eerst de constructor worden



aangeroepen. Een constructor voegt aan elk object een pointer toe die naar een speciale tabel, de Virtual Method Table (VMT), wijst. Deze tabel bevat de adressen van alle methoden die virtueel zijn. Wordt zo'n methode aangeroepen, dan wordt uit de VMT eerst het juiste adres gehaald, zodat de goede methode voor die instantie wordt aangeroepen. Dit gebeurt dus allemaal tijdens de executie van het programma en niet tijdens de compileerfase van het programma. Merk op dat een constructor nooit virtueel kan zijn, omdat er bij aanroep van de constructor nog geen VMT is, terwijl die dan wel nodig zou zijn om de juiste constructor te kiezen. Een andere beperking die aan het werken met virtuele methoden wordt gesteld is dat het aantal, type en volgorde van de parameters van de methode identiek moeten zijn (in het voorbeeld zijn er geen parameters, dus aan de eis is voldaan). Wanneer deze eis niet gesteld zou worden, was de compiler niet in staat om een foutmelding te geven indien een methode met de verkeerde parameters zou worden aangeroepen. Nogmaals, pas tijdens de executie van het programma is bekend welke methode wordt uitgevoerd.

Virtuele methoden in QuickPascal

Tot nu toe hebben we ons steeds beperkt tot de implementatie van OOP in Turbo Pascal, maar ook Microsoft heeft een Pascalcompiler die OOP ondersteund, QuickPascal. In QP is elke methode virtueel, wat inhoudt dat er geen overschrijving van methoden mogelijk is met andere parameters, omdat dat niet kan bij virtuele methoden. Wilt u dat toch, dan dient u de methode een andere naam te geven!

Destructors

De tegenhanger van een constructor is de destructor. De naam impliceert het al, met deze speciale procedure wordt een object weer verwijderd. Een object kan meerdere destructors bezitten, in tegenstelling tot het aantal constructors, dat ten hoogste één is. Bovendien kan een destructor virtueel zijn, aangezien de VMT al aangemaakt is door de constructor. Let er op dat een destructor pas kan worden aangeroepen nadat het object met een constructor is aangemaakt. Dit geldt tevens voor aanroepen van virtuele methoden (en dus ook methoden die een virtuele methode aanroepen). Door het compilerdirective {$R+} aan te zetten wordt deze fout door Turbo Pascal herkend. Een goede gewoonte is dan ook om voordat u ook maar iets doet met een instantie eerst de constructor te laten aanroepen. In de constructor bevinden zich dan ook meestal initialisaties. De algemene naam voor de constructor is 'Init' of 'Initialize', maar zoals u in bovenstaande source ziet kan het ook anders. De algemene naam voor een destructor is 'Done'.

Dynamische objecten

Vooralsnog zijn alleen statische objecten de revue gepasseerd, maar net als bij records en andere datastructuren is het ook mogelijk om dynamische objecten te creëren. Het grote voordeel is natuurlijk een efficiënter geheugengebruik. Een dynamisch object wordt in Turbo Pascal net zo aangemaakt als bij andere pointers, met 'New'.

```
Type Object1 = Object
       X : Integer;
       Constructor Init;
       Procedure WriteIt;Virtual;
     End;

     Object2 = Object(Object1)
       Procedure WriteIt;Virtual;
       End;

Var    O1 : ^Object1;
       O2 : ^Object2;

Constructor Object1.Init;
Begin
 X:=3;
 WriteIt;
End;
```

```
Procedure Object1.WriteIt;
Begin
 WriteLn (X);
End;

Procedure Object2.WriteIt;
Begin
 WriteLn (X*X);
End;


Begin
 New(O1);
 O1.Init;
 New(O2,Init);
 If O2  Nil Then WriteLn ('creëren
gelukt');
End.

Uitvoer                  : 3
                           9
                           creëren gelukt
Uitvoer zonder virtuele
functies                 : 3
                           3
                           creëren gelukt
```

U ziet dat O1 wordt aangemaakt door eerst met New(O1) ruimte op de heap te reserveren en daarna de constructor aan te roepen. Dan wordt dus pas de VMT voor O1 aangemaakt (En voor alle nog komende instanties van hetzelfde type als O1). Alleen bestaat de VMT-Pointer, een pointer die wijst naar het adres van de VMT, nog niet voor die andere instanties, dus moeten deze nog steeds met de constructor worden aangemaakt. U ziet echter dat de instantie O2 op een andere manier wordt aangemaakt. Het aanroepen van de constructor en de opdracht om ruimte op de heap te krijgen zijn hier in één statement samengevoegd. Deze schrijfwijze heeft een aantal voordelen waar we hier niet verder op in zullen gaan, behalve dan dat het nu heel makkelijk wordt om te kijken of het creëren van een instantie gelukt is. Als het niet gelukt is wijst O2 namelijk naar 'Nil', vandaar het testen daarop. Wanneer O2 een statisch object was, was dat na te gaan door de volgende Booleaanse functie uit te voeren:

```
If O2.Init Then WriteLn
              ('creëren gelukt');
```

Deze schrijfwijze doet een beetje vreemd aan, omdat Init hier als een functie beschouwd wordt die een waarde teruggeeft, terwijl het ook een speciaal soort procedure is. Dat komt omdat in constructors de instructie 'Fail' kan worden teruggegeven. Deze doet in feite hetzelfde als 'Exit', maar geeft ook aan dat de creatie van de instantie mislukt is. Wanneer in de constructor bijvoorbeeld een stuk geheugenruimte wordt gereserveerd en de heap dit niet beschikbaar heeft, dan kan dit door een Fail terug te geven doorgegeven worden.

Polymorfie

De combinatie van virtuele methoden en dynamische objecten stelt ons in staat om een de derde steunpijler van OOP te verwezenlijken, namelijk polymorfie. Polymorfie betekent letterlijk veelvormigheid, hetgeen in dit verband met het volgende voorbeeld verduidelijkt wordt. Stel dat u een basisobject DataType definieert. Deze krijgt een aantal virtuele methoden, waaronder de methode Print. Wanneer u nu afgeleide objecten maakt, zoals een String en een Integer, dan kunt u deze dynamisch aanmaken. Wordt nu de procedure WWrite gedefinieerd die als parameter een pointer naar een datatype meekrijgt, dan kan in die procedure een aanroep gedaan worden naar de Print-methode van dat specifieke object. Vergelijk dit eens met de normale WriteLn die TP bezit. U kunt elk datatype meesturen zonder dat u zich druk hoeft te maken over dat type. WriteLn bekijkt zelf wat het type is en onderneemt dan de juiste stappen.

```
Program PolyTest;
Uses Crt;

Type PDataType   = ^DataType;
     DataType    = Object
           Constructor Init;
           Procedure Print;Virtual;
          End;

     PStringType = ^StringType;
     StringType = Object(DataType)
           s : String;
           Procedure Print;Virtual;
          End;

     PIntType    = ^IntType;
     IntType     = Object(DataType)
           i : Integer;
           Procedure Print;Virtual;
          End;


Constructor DataType.Init;
Begin
End;

Procedure DataType.Print;
Begin
 WriteLn('Empty Type');
End;

Procedure StringType.Print;
Begin
 WriteLn(s);
End;
```

```
Procedure IntType.Print;
Begin
 WriteLn(i);
End;

Procedure WWrite(HP : PDataType);
Begin
 HP^.Print;
End;

Var DT : PDataType;


Begin
 ClrScr;
 New(DT,Init);
 WWrite(DT);
 New(PStringType(DT),Init);
 PStringType(DT)^.s:='dit is een string';
 WWrite(DT);
 New(PIntType(DT),Init);
 PIntType(DT) ^.i:=1000;
 WWrite(DT);
End.
```

De uitvoer van dit programma zal dan ook zijn:

```
Empty Type
dit is een string
1000
```

In de volgende DOS-Special zal een uitgebreider voorbeeld van polymorfisme ter sprake komen.

Voorbeeldprogramma

Het voorbeeldprogramma bestaat voor het grootste deel uit een file-finder die het mogelijk maakt om op meerdere disks in alle directories files te zoeken. Op de commandline wordt een mask meegegeven, waarbij wildcards zijn toegestaan. Indien de optie /h wordt meegegeven, dan wordt op alle drives vanaf c: gezocht, indien de optie /f wordt meegegeven op alle drives (dus vanaf a:). Wanneer geen extra optie wordt meegegeven wordt alleen de huidige drive afgewerkt. Een extra optie /q zorgt ervoor dat er gezocht wordt zonder dat de gevonden files op het scherm komen te staan. Bij een filefinder is dit niet zo handig, maar in afgeleide objecten echter wel. Het programma stopt niet wanneer een drive niet bestaat, maar zoekt verder naar de volgende drive. Elke keer wanneer een file voldoet aan het mask wordt de virtuele methode 'DoWithFile' aangeroepen, die in het geval van de filefinder de naam en de betreffende directory op het beeldscherm zet. Het gebruik van virtuele functies is in dit geval nog niet noodzakelijk, maar wel wanneer we door middel van inheritance een programma ontwikkelen dat kijkt hoeveel ruimte bepaalde bestanden in beslag nemen. In principe is doet dit hetzelfde als het programma 'used' uit de vorige DOS-Special, alleen is deze versie dus geschikt voor het zoeken op meerdere schijven. Diegenen die meerdere harddisks in de computer hebben zitten of nog met een oudere versie dan 4.xx van DOS werken bezitten nu dus een uitbreiding. Indien u zelf een keer een applicatie ontwikkeld waarbij het voorkomt dat u subdirectories op één of meerdere schijven moet doorzoeken, dan heeft u aan de filefinder een aanzet. Door het overschrijven van de methode 'DoWithFile', die in 'Search' wordt aangeroepen, kunt u uw eigen bewerkingen doen. De informatie over de file zit opgeslagen in de datamember 'FileInfo', die van het voorgedefinieerde type 'SearchRec' is. Informatie hierover vindt u bij de uitleg van de unit 'dos'. Het handigst is wanneer u aelf een nieuwe methode 'DoSearchingFor' ontwikkeld, vanwaaruit 'Search' wordt aangeroepen. In 'DoSearchingFor' kunt u dan een aantal initialisaties doen en na 'Search' de gevonden resultaten op het beeldscherm zetten. Alle objecten zijn hier dus statisch, zodat er nog niet echt sprake is van polymorfie.

```
Program DosExtend;

{$I-}  {bij een I/O foutmelding stopt het
        programma niet}

Uses Dos,Crt;

Type FileFinder
          = Object
     SearchMask  : String;
     FileInfo,
     DirInfo     : SearchRec;
             SearchAllHD,
             SearchAll,
             ScannedDir,
             QuietMode   : Boolean;
             Files       : LongInt;
{* aantal files wordt door ff bijgehouden *}

             Constructor Init;
             Procedure   PrintDir;
             Procedure   ParamInit;  Virtual;
             Procedure   PrintUsage; Virtual;
     Procedure   DoWithFile; Virtual;
     Procedure   SearchDisk
                         (RecDirInfo :
SearchRec);
             Procedure   Search;
             Procedure   DoSearchingFor;
    End;

     SpaceSearcher
          = Object(FileFinder)
     SpaceUsed   : LongInt;

             Procedure   DoWithFile; Virtual;
             Procedure   DoSearchingFor;
    End;


Constructor FileFinder.Init;

Begin
 SearchMask:=ParamStr(1);
 DirInfo.Name:='\';
 SearchAllHD:=False;
 SearchAll:=False;
 QuietMode:=False;
 Files:=0;
 ParamInit;
End;

Procedure FileFinder.ParamInit;
Begin
 If ParamCount = 3
 Then
   If ParamStr(3) = '/q'
   Then
     QuietMode:=True
   Else
     PrintUsage;
 If (ParamCount In [2..3])
 Then
   If ParamStr(2) = '/h'
   Then
     SearchAllHD:=True
   Else
     If ParamStr(2) = '/f'
     Then
       SearchAll:=True
     Else
       If (ParamCount = 2) And
          (ParamStr(2) = '/q')
       Then
         QuietMode := True
       Else
         PrintUsage;
 If Not(ParamCount In [1..3])
 Then
   PrintUsage;
```

```
 WriteLn('Searching for all ',SearchMask,
         ' files');
End;

Procedure FileFinder.PrintUsage;

Begin
 WriteLn('Gebruik : searchmask [/h] [/f]
([/q])');
 WriteLn('Opties  :');
 WriteLn('/h : alle harddisks');
 WriteLn('/f : harddisks + floppydisks');
 WriteLn('/q : quiet mode');
 Halt;
End;

Procedure FileFinder.SearchDisk
            (RecDirInfo : SearchRec);

{* deze recursieve procedure wordt eerst aange-
   roepen met '\', zodat vanaf de root van de
   disk begonnen wordt met zoeken.           *}

Var Attrib          : Word;
    Continue        : Boolean;
    F               : File;

Begin
  ChDir(RecDirInfo.Name);
  ScannedDir:=False;
  FindFirst(SearchMask,Archive,FileInfo);
  While DosError = 0 Do
    Begin
      DoWithFile;       {aanroep naar virt.meth.}
      ScannedDir:=True; {eerste file in deze dir}
      FindNext(FileInfo);
    End;
  FindFirst('*.*',Directory,RecDirInfo);
  While (RecDirInfo.Name[1] = '.') And
(DosError = 0) Do
    FindNext(RecDirInfo); { skip '.' en '..' }
  While (DosError = 0)  Do
   Begin
    Assign(F,RecDirInfo.Name);
    GetFAttr(F,Attrib);
    If Attrib = Directory Then
     Begin
      SearchDisk(RecDirInfo); { recursieve }
      ChDir('..');            { aanroep    }
     End;
    FindNext(RecDirInfo);
   End;
End;

Procedure FileFinder.Search;

Var DiskNumber,
    FromDisk,
    ToDisk       : Integer;
    ChangeTo,
    SaveDir,
    SaveDrive    : String;

Begin
 GetDir(0,SaveDrive);    {bewaar eerste drive}
 If SearchAllHD
 Then
  FromDisk:=3            {vanaf C:}
 Else
  FromDisk:=1;           {vanaf A:}
 If SearchAll Or SearchAllHD
 Then
  ToDisk:=26             {tot en met Z:}
 Else
  ToDisk:=1;             {alleen huidige disk}
 For DiskNumber:=FromDisk To ToDisk Do
  Begin
   If SearchAll Or SearchAllHD
   Then
    Begin
```

```
      ChangeTo:=(Chr(DiskNumber+64)) + ':';
      ChDir (ChangeTo);   {verander naar drive}
     End;
    GetDir(0,SaveDir);    {bewaar directorie}
    ChDir('\');           {ga naar root}
    If IOResult = 0 Then
     SearchDisk(DirInfo);{ga naar root en zoek}
    ChDir(SaveDir);       {herstel dir}
  End;
 ChDir(SaveDrive);        {herstel drive}
End;

Procedure FileFinder.DoWithFile;

{ deze virtuele methode kan vanuit andere
  DoWithFile methoden worden aangeroepen om de
  naam van de file en eventueel de directory
  te printen. ook wordt het aantal gevonden files
  bijgehouden. roep aan dmv
FileFinder.DoWithFile,
  dus niet alleen DoWithFile
  (zie SpaceSearcher.DoWithFile)
}

Begin
 If Not(QuietMode)
 Then
  Begin
   If Not(ScannedDir)
   Then
     PrintDir;
   WriteLn('      ',FileInfo.Name);
  End;
 Inc(Files);
End;

Procedure FileFinder.PrintDir;

Var Name : String;

Begin
 GetDir(0,Name);
 WriteLn (Name);
End;

Procedure FileFinder.DoSearchingFor;

Begin
 Search;
 WriteLn ('aantal files : ',Files)
End;

Procedure SpaceSearcher.DoWithFile;

Begin
 SpaceUsed:=SpaceUsed + FileInfo.Size;
 FileFinder.DoWithFile;
End;

Procedure SpaceSearcher.DoSearchingFor;

Begin
 SpaceUsed:=0;
 Search;
 WriteLn ('totale ruimte gebruikt door '
          ,SearchMask,' : ',SpaceUsed,' bytes');
 WriteLn ('aantal files : ',Files);
End;

{* hoofdprogramma, nu ingesteld op een
   SpaceSearcher, door type van SearchFor te
   veranderen in FileFinder wordt het een
   FileFinder. Door de smart-linker wordt
   onnodige code niet meegecompileerd
*}

Var SearchFor : SpaceSearcher;

Begin
  SearchFor.Init;
  SearchFor.DoSearchingFor;
End.
```

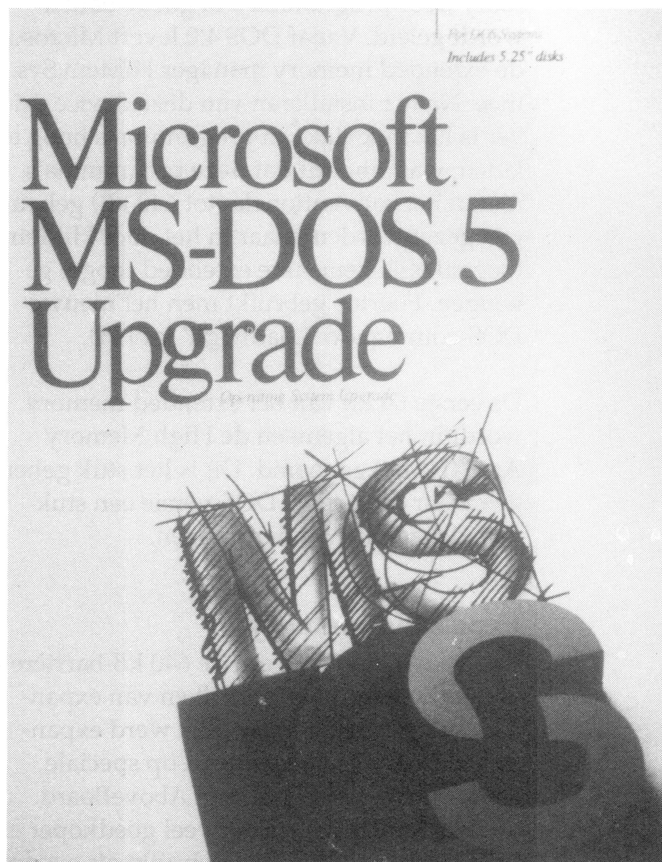
MS-DOS 5.0

Op 11 juni introduceerde Microsoft versie 5.0 van het besturingssysteem MS-DOS. Deze versie bevat vele verbeteringen op de voorgaande versies, waarvan we de belangrijkste in dit artikel zullen behandelen.

Veel aandacht is besteed aan de geheugenproblematiek die onlosmakelijk met MS-DOS is verbonden. Zo is het nu mogelijk om met een nieuw commando 'LoadHigh'-programma's (inclusief een deel van het systeem zelf) in het hoge deel van het geheugen te laden, waardoor een groter gedeelte van de onderste 640 kB beschikbaar blijft. Verder is de DOS-Shell in een nieuw jasje gestoken, en werd een aantal nieuwe mogelijkheden toegevoegd zoals het 'swappen' van programma's. De oude getrouwen Edlin en Basic zijn in DOS 5 vervangen door moderne versies, te weten de full-screen editor Edit en de QuickBasic interpreter. Ook wordt in de nieuwe DOS wat meer aandacht besteed aan het geven van on-line hulp-informatie over commando's. Tenslotte wordt nu een aantal handige utilities meegeleverd, waarvan enkele al uit het public domain-circuit bekend waren.

Het geheugen van de PC onder MS-DOS

Het is algemeen bekend dat MS-DOS niet probleemloos gebruik kan maken van al het geheugen in uw PC. Zoals bekend mag worden verondersteld, kunnen computers die gebaseerd zijn op de 8088/8086-processors slechts 1024 kiloByte ofwel 1 MegaByte adresseren. Oorspronkelijk werd deze hoeveelheid geheugen in twee gelijke stukken gedeeld: 512 kB voor gebruikersprogramma's en 512 kB voor het BIOS en hardware devices. Gelukkig werd kort daarna deze verhouding bijgesteld tot 640 kB voor programma's en 384 kB voor de rest (upper memory genoemd); dit is waar de bekende 640 kB-grens vandaan komt. Met de komst van de 80286- en 80386-processoren werd de hoeveelheid adresseerbaar geheugen vergroot tot vele Megabytes; het direct toegankelijke



geheugen bleef onder MS-DOS echter nog steeds beperkt tot die grens van 640 kB. Dit is eveneens het geval in versie 5.0 van MS-DOS.

Andere besturingssystemen zoals UNIX en OS/2 kunnen wel al het PC-geheugen direct aanspreken; tevens biedt Windows 3.0 in enhanced mode de mogelijkheid deze grens te omzeilen. Omdat de meeste PC-programma's echter onder MS-DOS draaien, werd een aantal hulpmiddelen ontwikkeld om het extra geheugen te kunnen gebruiken, hetgeen we nu zullen behandelen. De figuur geeft schematisch aan hoe het geheugen op een PC in elkaar steekt.

Extended memory

Een methode om van meer geheugen gebruik te kunnen maken, is het installeren van extended geheugen. Om dit geheugen onder MS-DOS toegankelijk te maken moet bovendien een extended memory manager geïnstalleerd worden. Een extended memory manager zorgt dat het gebruik van extended memory door programma's in goede banen wordt geleid. Vanaf DOS 4.0 levert Microsoft de extended memory manager HiMem.Sys mee. Na het installeren van deze device driver is het mogelijk om programma's hoog te laden, wat inhoudt dat deze programma's niet in het conventionele (tot 640 kB) geheugen gezet worden, maar in het door HiMem toegankelijk gemaakte extended (hoge) geheugen. Hiertoe gebruikt men het nieuwe DOS-commando 'LoadHigh' (of LH).

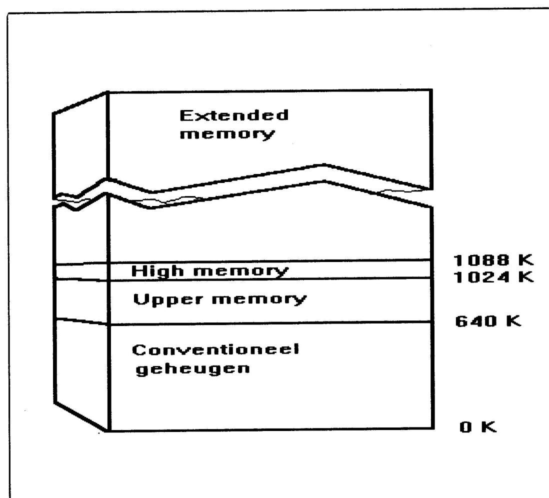
De eerste 64 kB van het extended memory wordt in het algemeen de High Memory Area (HMA) genoemd. Dit is het stuk geheugen waar de nieuwe DOS-versie een stuk van zichzelf kan onderbrengen.

Expanded memory

Een andere methode om de 640 kB-barrière te doorbreken is het gebruiken van expanded memory. Oorspronkelijk werd expanded memory veelal geleverd op speciale EMS-kaarten zoals het Intel AboveBoard. Tegenwoordig kan echter veel goedkoper extended memory worden gebruikt als expanded memory met behulp van speciale expanded memory managers zoals QEMM en EMM386 van Microsoft.

- DosKey
- Edit
- QBasic
- Help
- LoadHigh (lh)
- Mirror
- Undelete
- Unformat
- SetVer

Tabel 1. Nieuwe DOS-commando's



De geheugenindeling van de PC onder DOS

Nieuwe commando's

Deze nieuwe DOS-versie bevat zowel een aantal nieuwe commando's als uitbreidingen op bestaande commando's. We hebben reeds het commando 'LoadHigh' besproken, waarmee TSR-utilities en andere programma's 'hoog' geladen kunnen worden. Tabel één geeft een overzicht van de nieuwe DOS-commando's.

DosKey

DosKey is een programma waarmee het editen van de command-line mogelijk wordt; bovendien houdt dit programma een history-list van de laatste commando's bij. Velen zullen soortgelijke programma's al vanuit het public-domain circuit kennen. DosKey kan ook worden gebruikt om macro's te definiëren. De mogelijke command-line opties van DosKey zijn:

```
DosKey [/ReInstall] [/BufSize=size]
       [/History] [/Insert | /Overstrike]
       [/Macros]
```

De ReInstall-optie installeert een nieuw exemplaar van DosKey in het geheugen, waarbij de history-list wordt verwijderd. Met BufSize kan de buffergrootte worden ingesteld die DosKey gebruikt om commando's in te bewaren. Met /History wordt een overzicht gegeven van de commando's die in de buffer staan. De initiële modus (tussenvoegen of overschrijven) van DosKey kan met de opties Insert en Overstrike worden in-

gesteld. Met /Macros tenslotte wordt een overzicht gegeven van alle tot dusver gedefinieerde macro's.

Macro's ofwel aliases kunnen met DosKey als volgt worden gedefinieerd: type 'DosKey' in, gevolgd door de naam van de macro en de definitie van de macro. Namen van parameters kunnen met $1, $2 enzovoorts worden aangegeven. Zo kunnen we bijvoorbeeld de macro 'cat' definiëren als 'type':

```
DosKey cat=type $1
```

Nu zal 'Cat \AutoExec.Bat' hetzelfde resultaat opleveren als het Type-commando. Ook ingewikkelder macro's kunnen op deze manier worden gedefinieerd:

```
DosKey Move=Copy $1 $2 $t del $1
```

Hiermee zal 'Move' eerst kopiëren en vervolgens het source-bestand verwijderen. Het teken $t staat voor een scheiding van commando's. Dit is ook mogelijk op de commandline zelf: door Ctrl-T in te typen kunnen diverse commando's op één regel achter elkaar worden ingevoerd:

```
MkDir NewDir  cd NewDir
```

Edit en QBasic

Naast de bekende regel-editor 'Edlin' bevat DOS 5 ook een wat meer geavanceerde editor, 'Edit' genaamd. Dit programma ondersteunt het gebruik van de muis en maakt gebruik van pull-down menu's en vensters. Het Edit.Com-programma lijkt erg klein, maar maakt gebruik van QBasic.Exe, ongeveer 250 kB in grootte. Dit is de nieuwe Basic-interpreter, die voor Edit beschikbaar moet zijn. De Basic-interpreter is in DOS 5 dus vervangen door QuickBasic versie 1.0; hiermee kunnen echter geen .EXE-bestanden worden aangemaakt van Basic-programma's.

Help

Dit programma kan worden gebruikt om hulp-informatie op te vragen over de beschikbare MS-DOS commando's. Het intypen van HELP zonder parameters geeft een overzicht van deze commando's, inclusief een korte omschrijving. Om hulp te verkrijgen over het commando Dir kan 'HELP Dir' worden ingetypt. Ook kan overigens 'Dir /?' worden gebruikt om gedetailleerde informatie te verkrijgen; dit is bovendien sneller dan het gebruik van HELP.

Mirror

Het Mirror-programma slaat informatie op over schijven die later gebruikt kan worden bij de commando's UnDelete en UnFormat. Het geven van het commando 'Mirror /tc' bewaart een kopie van de File Allocation Table (FAT) en start een proces genaamd 'deletion tracking'. Een speciaal programma wordt hiertoe als een TSR in het geheugen geladen en houdt een lijst bij van bestanden die verwijderd worden.

Voorts kan Mirror een bestand maken met cruciale informatie over de huidige partities van de harde schijf. Dit bestand wordt op een diskette opgeslagen, en kan later door UnFormat gebruikt worden om een abusievelijke formattering van een schijf ongedaan te maken.

Mirror kan op de volgende manieren gebruikt worden:

```
Mirror [Drive:] [/1] /tTargetDrive
       [-Entries]
```

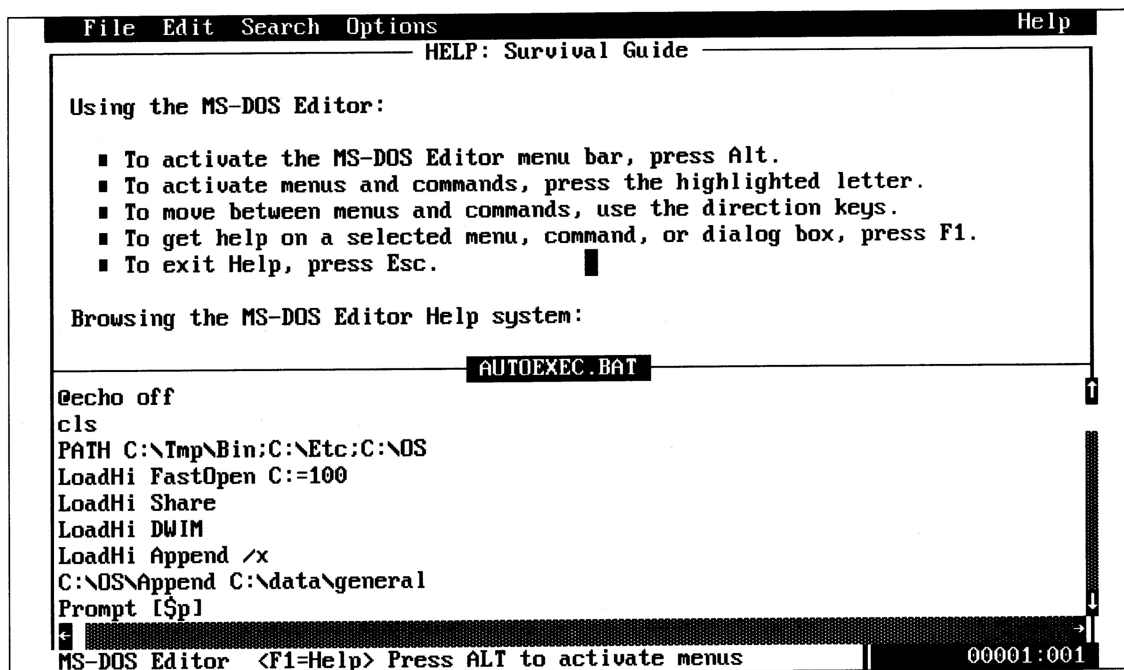
Hiermee wordt het 'deletion tracking'-systeem geïnstalleerd voor station 'TargetDrive'. Het informatie-bestand zal op 'Drive' worden opgeslagen. Met 'Entries' kan worden aangegeven hoeveel verwijderde bestanden maximaal worden bijgehouden. De optie /1 voorkomt dat Mirror bij het inlezen van de schijf-informatie een backup maakt van de vorige informatie.

```
Mirror /u
```

Hiermee kan Mirror uit het geheugen worden verwijderd, waarbij het 'deletion tracking'-proces gestaakt wordt. Voorwaarde is wel dat er geen TSR-programma's zijn geladen na Mirror.

```
Mirror /Partn
```

De MS-DOS Editor



Deze optie zal een bestand met partitie-gegevens van de harddisk op een diskette opslaan. Als er dan iets fout gaat met de partitionering van de harde schijf, kan het UnFormat-programma met behulp van dit bestand de zaken herstellen. Omdat de partitionering van een harde schijf zelden wordt gewijzigd, hoeft dit in de regel maar één maal te gebeuren.

UnDelete

Met UnDelete kunnen verwijderde bestanden worden teruggehaald. Als een bestand wordt verwijderd, bijvoorbeeld met het Del-commando, wordt alleen de entry van het bestand in de directory als 'gewist' gemarkeerd. Het eerste teken van de bestandsnaam gaat hierbij verloren, maar de feitelijke gegevens staan nog steeds op de schijf, totdat er nieuwe informatie overheen wordt geschreven. Met UnDelete kunnen gewiste bestanden worden teruggehaald; hierbij moet het eerste teken van de naam worden ingetoetst. Indien het 'deletion tracking'-systeem van Mirror is geïnstalleerd, is dit niet nodig.

Het UnDelete-proces gaat dan ook niet altijd goed (bijvoorbeeld als na het wissen nieuwe bestanden zijn aangemaakt), maar met Mirror is dit proces veiliger. Vier opties kunnen aan UnDelete worden meegegeven:

```
UnDelete [Filename] [/List | /All]
                    [/Dos | /DT]
```

De List-optie laat alleen zien welke gewiste bestanden teruggehaald kunnen worden, maar onderneemt verder geen actie. Met /All worden gewiste bestanden teruggehaald zonder dat er om een bevestiging wordt gevraagd bij elk bestand. Een #-teken wordt als eerste letter van de bestandsnamen gebruikt, tenzij het Mirror-programma geladen is.

De Dos-optie maakt gebruik van de directory-informatie die DOS altijd opslaat over gewiste bestanden, en maakt dus geen gebruik van de deletion tracking file. Met /DT tenslotte wordt niet gevraagd om een bevestiging, en wordt de informatie uit het door Mirror bijgehouden bestand gehaald

UnFormat

Het UnFormat-programma kan worden gebruikt om schijven die met Format geformatteerd zijn, te 'on-formatteren'. Bovendien kan UnFormat de partitie-informatie die door Mirror is opgeslagen gebruiken om de partitie-tabel van een harde schijf te herstellen. UnFormat kan op de volgende manieren gebruikt worden:

```
UnFormat [/j]
```

Hiermee kan worden gecontroleerd of het door Mirror aangemaakte bestand overeenkomt met de huidige status van de harde schijf.

```
UnFormat [/u] [/l] [/Test] [/p]
```

De /u-optie on-formatteert de harde schijf zonder gebruik te maken van de Mirror-informatie. De /l-optie laat bovendien tijdens het proces alle bestanden en directories zien die te voorschijn komen. Met /Test kan worden bekeken hoe een schijf hersteld zou worden door UnFormat, zonder dat dit ook werkelijk plaatsvindt. Met /p wordt alle uitvoer naar de printer gerapporteerd.

```
UnFormat [/Partn] [/l]
```

Met /Partn kan een beschadigde partitietabel worden hersteld, mits tevoren met Mirror /Partn een speciaal bestand op diskette is bewaard. De /l-optie in combinatie met deze optie laat alleen de partitie-gegevens zien.

SetVer

SetVer maakt het mogelijk om programma's die van een bepaalde DOS-versie afhankelijk zijn en bij het opstarten op die specifieke versie controleren, te laten denken dat ook werkelijk die DOS-versie gedraaid wordt. Hiertoe wordt een tabel bijgehouden met programma's en de corresponderende DOS versie, die gebruikt wordt bij het opstarten van de applicatie.

Het SetVer-programma wordt bij het opstarten door het Config.Sys in het geheugen geladen, waarbij een aantal programma's al in de lijst aanwezig is. Normaal gesproken zal het gebruik van SetVer niet noodzakelijk zijn.

Nieuwe mogelijkheden van bestaande commando's

Veel bestaande commando's zijn uitgebreid met een aantal nieuwe opties. Zoals reeds genoemd kan nu bij elk commando hulp-informatie worden opgevraagd door middel van de /?-optie. We zullen de meest interessante hier noemen.

Allereerst is het meest gebruikte DOS-commando, Dir, flink uitgebreid. Er kan nu met behulp van diverse opties worden gesorteerd, geselecteerd op bestandsattributen, en recursief gezocht worden naar bestanden. De opties zijn als volgt:

```
Dir [FileSpec] [/p] [/w] [/a:Attrib]
               [/o:SortOrder] [/s]
               [/b] [/l]
```

De eerste nieuwe optie is /a, waarmee naar files gezocht kan worden met bepaalde attributen. Attributen kunnen op de plaats van 'Attrib' ingevuld worden, en zijn bijvoorbeeld 'h' voor hidden of 'd' voor directories. Door een - teken tussen /a: en het attribuut te zetten wordt juist niet gezocht naar bestanden met dat attribuut.

Sorteren van de bestanden kan gebeuren met /o, waarna de sorteer-volgorde kan worden aangegeven. Een aardige sorteermogelijkheid is de letter 'g' waarmee directories eerst worden getoond, gevolgd door bestanden. Met '-g' is dit juist andersom. De /s-optie laat bovendien ook de bestanden in alle 'onderliggende' sub-directories zien. Met /b wordt 'bulk'-output verkregen; hierbij worden alleen de bestandsnamen getoond en geen overige informatie. Tenslotte worden met /l de bestandsnamen in kleine letters getoond.

Met behulp van de DOS-variabele DirCmd kunnen opties permanent worden ingesteld, zodat ze niet bij elk Dir-commando opnieuw ingetypt hoeven te worden. Door de volgende regel op te nemen in uw Autoexec.Bat-bestand, geeft een Dir-commando in het vervolg altijd een gesorteerde lijst van directories gevolgd door bestanden, waarbij alles in kleine letters wordt gezet:

```
Set DirCmd=/l/o:gn
```

Ook het Format-commando heeft twee nieuwe opties, te weten:

- **/q** Formatteert een disk door enkel de FAT en de root-directory weg te halen, waarmee een winst kan worden bereikt van 80% op de formatteertijd. Voorwaar-

```
                              MS-DOS Shell
 File  Options  View  Tree  Help
 C:\BIN\BC\CLASSLIB\INCLUDE
[A:]  [B:]  [C:]  [D:]  [I:]  [S:]  [X:]  [Y:]

         Directory Tree                C:\BIN\BC\CLASSLIB\INCLUDE\*.*
 [-] C:\                               ABSTARRY.H         8,224  02-13-91
  ├─[-] BIN                            ARRAY   .H         5,788  02-13-91
  │  ├─[-] BC                          ASSOC   .H         2,488  02-13-91
  │  │  ├─[ ] BGI                      BAG     .H         2,284  02-13-91
  │  │  ├─[ ] BIN                      CLSDEFS .H         2,486  02-13-91
  │  │  ├─[-] CLASSLIB                 CLSTYPES.H         5,843  02-13-91
  │  │  │  ├─[ ] EXAMPLES              COLLECT .H         3,401  02-13-91
  │  │  │  ├─[ ] INCLUDE               CONTAIN .H         5,369  02-13-91
  │  │  │  ├─[ ] LIB                   DBLLIST .H         9,273  02-13-91
  │  │  │  └─[ ] SOURCE                DEQUE   .H         3,856  02-13-91
  │  │  ├─[ ] DOC                      DICT    .H         3,325  02-13-91
  │  │  ├─[-] EXAMPLES                 DLSTELEM.H         2,012  02-13-91
  │  │  │  ├─[ ] STARTUP               HASHTBL .H         7,268  02-13-91
  │  │  │  └─[ ] TCALC                 LDATE   .H         2,853  02-13-91
  │  │  ├─[-] INCLUDE                  LIST    .H         5,523  02-13-91
  │  │  │  └─[ ] SYS                   LSTELEM .H         1,789  02-13-91
  │  │  └─[ ] LIB                      LTIME   .H        11,542  02-13-91

 F10=Actions  Shift+F9=Command Prompt                               9:09a
```

de is wel dat de disk reeds eerder geformatteerd is, en dat er geen bad sectors aanwezig zijn op de schijf;

° **/u** Formatteert een schijf, waarbij het onmogelijk is om later met unformat de toestand te herstellen. Deze optie kan bijvoorbeeld gebruikt worden om schijven te formatteren waarop tijdens het gebruik fouten zijn ontstaan, of waarop betrouwbare informatie heeft gestaan.

Vanaf deze versie van DOS worden ook de nog weinig gebruikte 2.88 MegaByte 3.5"-schijven ondersteund. Zodoende wordt bij de /f:-optie van het Format-commando ook dit formaat herkend.

Nieuwe Config.Sys-commando's

Drie nieuwe Config.Sys-commando's zijn beschikbaar in DOS 5: 'DOS' waarmee een gedeelte van het systeem 'hoog' geladen kan worden, 'DeviceHigh' waarmee hetzelfde met device-drivers kan worden gedaan, en 'Switches' waarmee een enhanced keyboard een standaard keyboard kan emuleren.

Het DOS-commando kan worden gebruikt om een gedeelte van het DOS in de high-memory area (HMA) te laden. Er wordt bovendien een verbinding gelegd tussen het 'lage' en het 'hoge' DOS, indien u de toevoeging 'UMB' gebruikt:

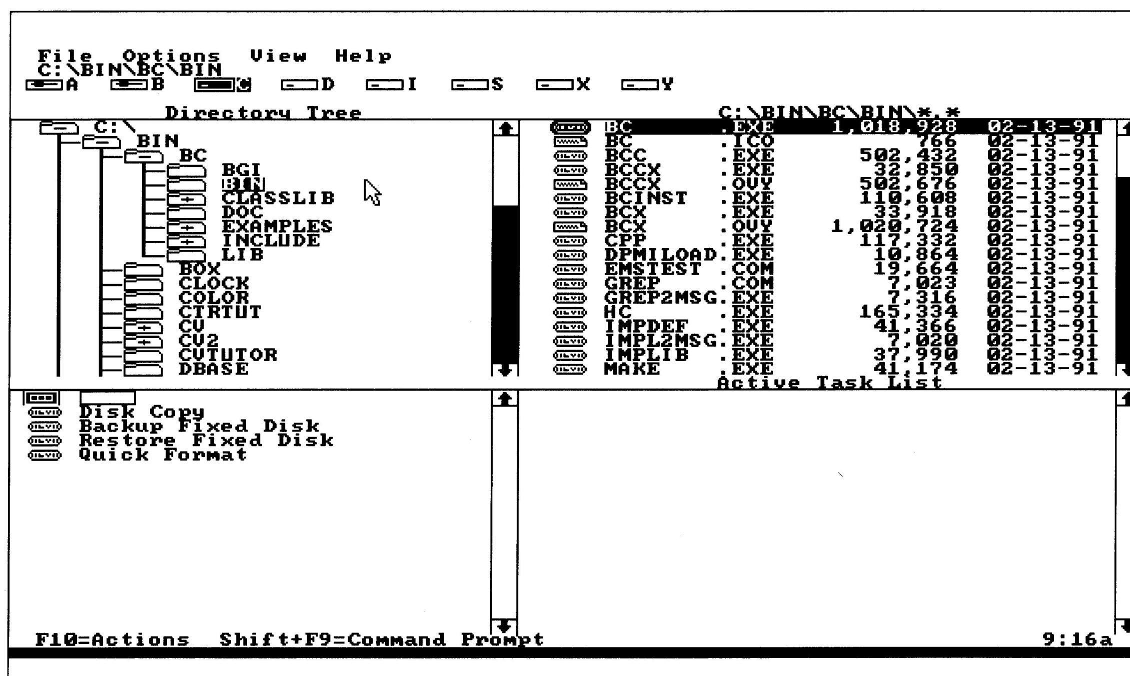
```
Device=HiMem.Sys
DOS=High,UMB
```

Voorwaarde is uiteraard dat de HiMem.Sys device-driver geladen is, die het hoog laden mogelijk maakt. Ter illustratie het verschil tussen het laag en hoog laden van DOS 5.0: door het 'Dos=High'-commando in het Config.Sys-bestand te zetten wordt de hoeveelheid beschikbaar conventioneel geheugen vergroot van 577 kB tot liefst 623 kB: een winst van 46 kB!

Het DeviceHigh-commando werkt hetzelfde als het Device-commando:

```
Device=HiMem.Sys
DOS=UMB
DeviceHigh=Ansi.Sys
```

Door gebruik te maken van dit commando kunnen device-drivers net als DOS hoog geladen worden, wat het conventionele geheugengebruik verder terugdringt.



MS DOS Shell in grafische mode

De DOS-Shell

Net als DOS 4.0 beschikt ook DOS 5.0 over een gebruikersinterface genaamd DOS-Shell. Dit programma beschikt over menu's, vensters en kan met de muis bestuurd worden. Vanuit DOS-Shell kan door directories worden gebladerd en kunnen programma's worden opgestart. Nieuw is de mogelijkheid om twee of meer programma's tegelijk in het geheugen te laden, waartussen gewisseld kan worden met Ctrl-Esc of Alt-Tab.

Beschikbaarheid

DOS 5.0 wordt meegeleverd met nieuwe computersystemen en is dus als zodanig niet los verkrijgbaar. Voor bezitters van oudere (vanaf versie 2.1) DOS-versies is een speciale upgrade-kit verkrijgbaar, waarmee het bestaande systeem wordt vervangen door versie 5. Voor bedrijven die in één keer alle DOS-computers willen upgraden naar versie 5.0 is een site-license agreement beschikbaar. Medio september wordt de Nederlands versie van DOS 5 uitgebracht.

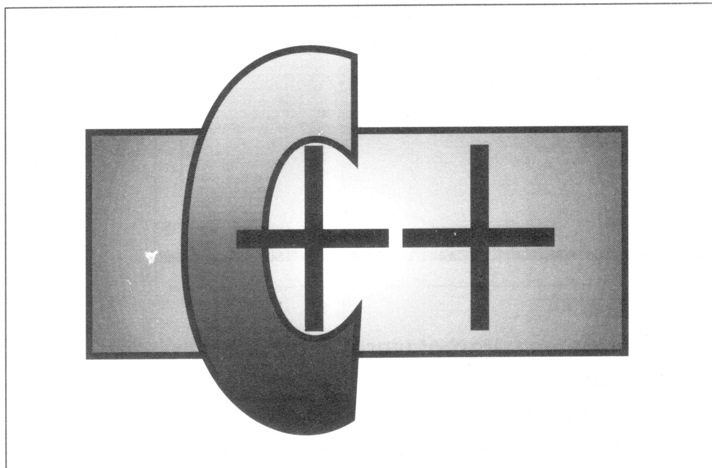
Conclusie

Het belangrijkste aan de nieuwe DOS-versie is waarschijnlijk de mogelijkheid om programma's, device-drivers en vooral een deel van het systeem zelf hoog te laden, waardoor de hoeveelheid beschikbaar conventioneel geheugen vergroot wordt. Voorts is een aantal commando's gestroomlijnd en verbeterd en de hulp-informatie is ook een welkome aanvulling. We hebben tot dusver geen problemen met bestaande programmatuur ondervonden; het SetVer-programma kan in zulke gevallen ook uitkomst bieden.

Besproken produkt: MS-DOS 5.0 Upgrade
Prijzen (excl. BTW):
f 229,-- voor de internationale versie
f 249,-- voor de Nederlandse versie
Informatie: Microsoft, Hoofddorp 02503-13181

Datastructuren in C++

In bijna elk programma worden ze wel gebruikt: datastructuren. Of het nu een gelinkte lijst van bestandsnamen betreft, een stack van waarden of een queue van jobs, steeds zal het vervelende werk van het schrijven van functies die betrekking hebben op deze datastructuren opnieuw moeten gebeuren. Dit artikel zal duidelijk maken hoe de voordelen van C++ kunnen worden aangewend om eenmaal geschreven code steeds opnieuw te kunnen gebruiken. Naast de object-georiënteerde techniek van data-hiding, wordt ook genericiteit behandeld, alsmede code hergebruik door middel van overerving.



Natuurlijk maken we voor een gelinkte lijst in C++ gebruik van de class. Programma's die een lijst gebruiken hebben immers niets te maken met de interne implementatie, maar gebruiken de gedefinieerde interface van member-functies. We zullen eerst kijken wat de data-members van de class zijn. Zoals in de listing is te zien, bevat een gelinkte lijst twee pointers (Last en Cursor) naar een structure Link. We houden de laatste Link bij, waarvoor geldt dat Last-Next gelijk is aan het eerste element, zoals in de definitie van First te zien is.

```
class List
{
  private:
    struct Link
    {
      eType e;
      Link *Next;
      Link(eType e) { e = p;
        Next = 0;
      }
    } *Last, *Cursor;
    int nElem;

  // ...
```

Tevens houden we in 'nElem' bij hoeveel elementen de huidige lijst heeft. We zien dat de Link structure binnen de List-class gedefinieerd wordt; dit houdt in dat Link buiten de scope van de List-class onzichtbaar is. Link heeft een data-element e en een pointer Next naar de volgende Link. Tevens heeft deze class een constructor, waarmee snel een nieuw Link-object kan worden aangemaakt.

De class List heeft twee constructors; de eerste neemt geen argumenten en wordt daarom de default constructor genoemd. Deze constructor initialiseert beide pointers Last en Cursor op nul, en zet bovendien de teller nElem op nul. De tweede constructor neemt een eType argument dat het eerste element aangeeft; de definitie van deze constructor staat niet in de class-definitie, maar in het bestand List1.cpp.

```
  // ...
  public:
    List( );
    List( eType );

    void AddHead( eType );
    void AddTail( eType );
    void Append( eType );
    eType GetHead( );

    int IsEmpty( );

    void Reset( );
    eType operator()();

    int Length( );
};
```

We kunnen elementen zowel aan de kop als aan de staart van de lijst toevoegen, respectievelijk met de functies AddHead en AddTail. Het synoniem Append is beschikbaar voor AddHead. Terugvragen van een element kan alleen geschieden aan de kop, met de functie GetHead. We zouden weliswaar een functie GetTail kunnen schrijven, maar deze zou vanaf het begin van de lijst

moeten zoeken naar het laatste element wat niet erg efficiënt is. De GetHead-functie verwijdert het element aan de kop van de lijst en geeft het terug aan de caller.

De functie IsEmpty kan gebruikt worden om te controleren of de lijst leeg is. De functie Length geeft de lengte van de huidige lijst, ofwel de waarde van de variabele nElem. Een bijzondere vermelding verdient de zogenaamde iterator; deze functie, in de List-class geïmplementeerd als de operator(), geeft bij elke aanroep het volgende element in de lijst terug. Wordt het einde bereikt, dan geeft de iterator eenmaal nul terug, waarna weer van voren af aan wordt begonnen. Tussentijds kan de Reset-functie gebruikt worden om de aanwijzer naar de kop van de lijst te laten wijzen. Met een iterator kan dus eenvoudig door een lijst worden gewandeld.

Generieke lijsten

Genericiteit betekent dat een datatype (zoals een gelinkte lijst) in één keer gedefinieerd kan worden voor alle datatypen. Helaas biedt de huidige versie van C++ nog niet de mogelijkheid van genericiteit zoals Ada en Eiffel dat bieden, maar zal dit pas in de volgende versie mogelijk zijn. We moeten ons voorlopig dus behelpen met een lapmiddel.

Maar we zitten niet zover van een generieke List-class af; we hebben immers eType als pointer naar void (void*) gedefinieerd, zodat alle andere pointers naar dit type kunnen worden geconverteerd. Stel dat we een structuur voor persoonsgegevens hebben, die er als volgt uitziet:

```
struct Person
{
  char *Name;
  int Age;
};
```

Nu kunnen we met inheritance (overerving) een nieuw datatype aanmaken, PersonList, dat gebaseerd is op een List maar rekening houdt met het Person type:

```
class PersonList : public List
{
  public:
    void AddHead( Person *p )
      { List::AddHead( (void*)p ); }
    Person *GetHead( )
      {
        return (Person*)
               List::GetHead( );
      }
    // ...
};

PersonList p;
```

We zien dus dat de member-functies niets anders doen dan een cast verzorgen van Person* naar void*, vice versa. Verder hoeven we niets te doen, wat op zich al een behoorlijke tijdswinst kan opleveren. Maar we kunnen dit proces automatiseren: door slim gebruik te maken van de pre-processor, kunnen we met enkele regels steeds een nieuw, afgeleid datatype introduceren dat van List afgeleid wordt. In dat geval zou een nieuwe class als volgt worden gedefinieerd:

```
typedef Person *PersonPtr;
gListDeclare(PersonPtr);

gList(PersonPtr) p;
```

Hierbij wordt gebruik gemaakt van twee macro's gListDeclare en gList, respectievelijk om een nieuw type te introduceren en een variabele van dat type aan te maken. De gListDeclare-macro maakt gebruik van de preprocessor-mogelijkheid om met ## twee strings aan elkaar te plakken (te stringizen), en doet in feite hetzelfde als we handmatig in het bovenstaande voorbeeld hebben gedaan:

```
#define gListDeclare(Type) \
class gList##Type : public List \
{ \
  public: \
    void AddHead(Type t) \
      { List::AddHead((void*)t); }  \
    Type GetHead() \
      { return Type(List::GetHead()); }\
};
```

Nu zal gListDeclare(PersonPtr) dus expanderen tot 'class gListPersonPtr : public List' enzovoort. Ofwel, die ene regel zorgt ervoor dat een compleet nieuw type wordt geïntroduceerd. Dit is de beste simulatie van genericiteit die we op dit moment in C++ kunnen realiseren. De generieke lijst is te vinden in het bestand glist1.h.

Code-hergebruik

We hebben nu dus een complete, werkende class voor een gelinkte lijst. We kunnen elementen toevoegen aan de voor- en achter-

kant, en elementen weghalen. Nu willen we een stack en een queue gaan implementeren, die echter intern veel gemeen hebben met een gelinkte lijst. Een stack heeft de AddHead- en GetHead-functies van een lijst, waarbij ze Push en Pop genoemd worden. Een queue heeft de functies AddTail en GetHead, respectievelijk als EnQueue en DeQueue. Gelukkig biedt C++ ons de mogelijkheid stacks en queues te ontwerpen zonder veel moeite, en wel door ze beide van List af te leiden. Hierbij maken we gebruik van private inheritance; dit houdt in dat de members van de basis-class 'vergeten' worden, ze kunnen dus niet gebruikt worden met de afgeleide class. Dit wordt gedaan om een nieuwe interface te kunnen definiëren. Functies van List die we wel willen gebruiken in de afgeleide class, moeten we expliciet noemen. De Stack-definitie wordt nu als volgt:

```
class Stack : private List
{
  public:
    // ...
    void Push( eType p )
      { AddHead(p); }
    eType Pop()
      { return GetHead(); }

    List::IsEmpty;
};
```

De definitie van een Queue-class gaat analoog. In de Stack-class wordt met 'List::IsEmpty' aangegeven dat van de List-class de IsEmpty-functie wel deel uitmaakt van de Stack-interface. Dus de functies AddHead, AddTail enzovoort kunnen niet gebruikt worden met een Stack, maar IsEmpty wel.

Uit de voorbeelden van dit artikel blijkt dat C++ grote mogelijkheden biedt ten aanzien van code-hergebruik, een van de sterke punten van object-georiënteerd programmeren. De sources zijn geschikt voor de Turbo en Borland C++ compilers, de meest populaire C++ compilers voor MS-DOS. Eveneens zijn ze getest met diverse andere C++ compilers.

```
// List1.h
// Linked list class

#if !defined LIST_H
#define LIST_H

// List
//
// Methods:
//   AddHead
//   AddTail == Append
//   GetHead
//   GetTail
//   IsEmpty
//

typedef void *eType;
```

```
class List
{
  private:
    struct Link
    {
      eType e;
      Link *Next;
      Link(eType p) { e = p; Next = 0; }
    } *Last, *Cursor;
    int nElem;

  public:
    List( ) { Last = Cursor = 0; nElem = 0; }
    List( eType );
    ~List( );

    // functions to add or remove items

    void AddHead( eType );
    void AddTail( eType );
    void Append( eType e ) { AddTail(e); }
    eType GetHead();

    int IsEmpty() { return Last==0; }

    void Reset();    // Reset iterator
    eType operator()();// iterator

    int Length() { return nElem; }
};
#endif

// List1.cpp
// list class methods

#include "..\ADT\List1.h"

#define First Last-Next

List::List( eType p )
{
  Last = new Link(p);
  First = Last;
  nElem = 0;
  Reset();
}

  // Destructor
List::~List( )
{
  Link *p = Last, *q = p;
  while ( q )
  {
    q = p->Next;
    delete p;
  }
}

  // List::AddHead
  // Add a new element to the head of the list
void List::AddHead( eType p )
{
  Link *nw = new Link(p);
  if ( Last )      // list is not empty
  {
    nw->Next = First; // insert before head
    First = nw;       // make circular
  }
  else
  {
    Last = nw;
    First = Last;
  }
  nElem++;
  Reset();
}

  // List::AddTail
  // Add a new element to the tail of the list
void List::AddTail( eType p )
```

```
{
  if ( IsEmpty() )
    AddHead(p);
  else
  {
    Link *nw = new Link(p);
    nw->Next = First;
    Last = First = nw;
    nElem++;
    Reset();
  }
}

  // List::GetHead
  // Get & remove element at head of list
eType List::GetHead()
{
  if ( IsEmpty() )
    return 0;

  Link *get = First;
  eType p = get->e;

  if ( get == Last )
    Last = 0;
  else
    First = get->Next;

  delete get;

  nElem--;
  return p;
}

  // List::operator()
  // Walk through the list, returning the
  // next element at each subsequent call
void *List::operator()()
{
  Link *p = Cursor;

    // determine next iteration value
  if ( Cursor && Cursor == Last )
    Cursor = 0;
  else if ( Cursor == 0 )
    Cursor = First;
  else
    Cursor = Cursor->Next;

  return p->e;
}

  // List::Reset
  // Reset iterator
void List::Reset()
{
  Cursor = First;
}

// Stack1.h

#if !defined STACK1_H
#define STACK1_H

#include "..\ADT\List1.h"

class Stack : private List
{
  public:
    Stack() { }
    Stack( eType p ) : List(p) { }

    void Push( eType p ) { AddHead(p); }
    eType Pop() { return GetHead(); }

    List::IsEmpty;

    int Depth() { return Length(); }
};
#endif
```

```
// gStack1.h
// Generic stack definition

#if !defined GSTACK_H
#define GSTACK_H

#include "..\ADT\Stack1.h"

#define gStack(Type) gStack##Type

#define gStackdeclare(Type)          \
  class gStack##Type : public Stack    \
  {                                    \
    public:                            \
      void Push(Type t)                \
        { Stack::Push(eType(t)); }     \
      Type Pop()                       \
        { return Type(Stack::Pop()); } \
  }
#endif

// Queue1.h

#if !defined QUEUE_H
#define QUEUE_H

#include "..\ADT\List1.h"

// Queue
//
// Methods:
//   EnQueue
//   DeQueue
//   IsEmpty

class Queue : private List
{
  public:
    Queue() { }
    Queue( eType p ) : List(p) { }

    void EnQueue( eType p )
      { List::AddTail(p); }
    eType DeQueue()
      { return List::GetHead(); }

    List::IsEmpty;
};
#endif

// gQueue1.h
// Generic queue definition

#if !defined GQUEUE_H
#define GQUEUE_H

#include "..\ADT\Queue.h"

#define gQueue(Type) gQueue##Type

#define gQueuedeclare(Type)                  \
  class gQueue##Type : public Queue          \
  {                                          \
    public:                                  \
      void EnQueue(Type t)                   \
        { Queue::EnQueue(eType(t)); }        \
      Type DeQueue()                         \
        { return Type(Queue::DeQueue()); }   \
  }
#endif
```

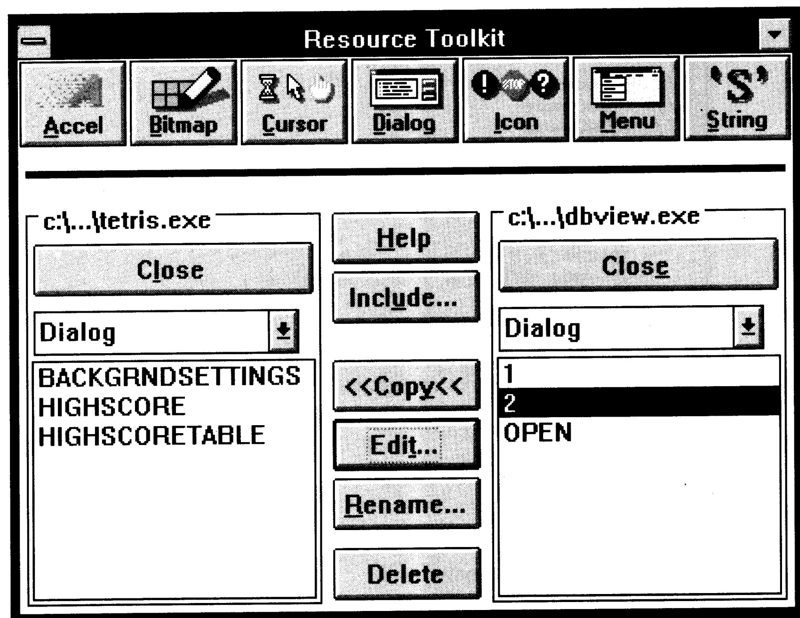
Programmeren in Windows

deel 2

In de vorige DOS Special hebben we de grondbeginselen van het Windows-programmeren behandeld. Er werd gesproken over de algemene opbouw van een Windows-programma en een overzicht gegeven van de verschillende soorten resources. We zullen in dit artikel wat dieper ingaan op het implementeren van enkele technieken, die voor Windows-programma's kenmerkend zijn. De veelgebruikte dialog box zullen we behandelen, alsmede diverse controls.

Resources

Zoals in het vorige artikel al te lezen is, zijn de resources van een Windows-programma



De Whitewater Resource Toolkit

de bitmaps, menu's, fonts, icons etcetera waar het programma gebruik van maakt. Resources kunnen handmatig in een .RC-bestand worden aangemaakt, maar het is gemakkelijker gebruik te maken van een speciale resource toolkit. Sommige resources zoals icons en mouse cursors kunnen alleen met een dergelijk hulpprogramma worden aangemaakt. We zullen ons in dit artikel beperken tot het definiëren en het gebruiken van dialog boxes.

Dialog Boxes

Dialog boxes zijn popup windows die een programma gebruikt om te communiceren met de gebruiker. Een bekende dialog box is de About box die in vrijwel geen enkel programma ontbreekt en die informatie geeft over de versie van het programma, de auteur enzovoorts.

Er zijn twee soorten dialog boxes: Modal en Modeless. Modal is de meest eenvoudige versie, en is het type van veel About boxes. Het verschil tussen Modal en Modeless dialog boxes is dat de eerste soort van de gebruiker verlangt dat hij eerst het dialoog afmaakt voordat met het programma verder kan worden gegaan. Modeless dialog boxes leiden hun eigen leven, zodat zo'n box niet afgesloten hoeft te worden alvorens met het hoofdprogramma verder gegaan kan worden.

Het aanmaken van een Modal dialog box

De definitie van een Modal dialog box staat in een resource-bestand. Deze definitie bevat markeringen voor zogenaamde controls; dit zijn objecten zoals tekst, list boxes en push buttons. Een bekende dialog box is het window dat getoond wordt om een bestand te openen. Hierin is naast tekst een list box te vinden met de beschikbare bestandsnamen alsmede push buttons voor 'OK' en 'Cancel'. De volgende definitie zou dan in een .RC-bestand komen te staan:

```
Open DIALOG 10, 10, 150, 120
STYLE WS_MODALFRAME | WS_SYSMENU
BEGIN
```

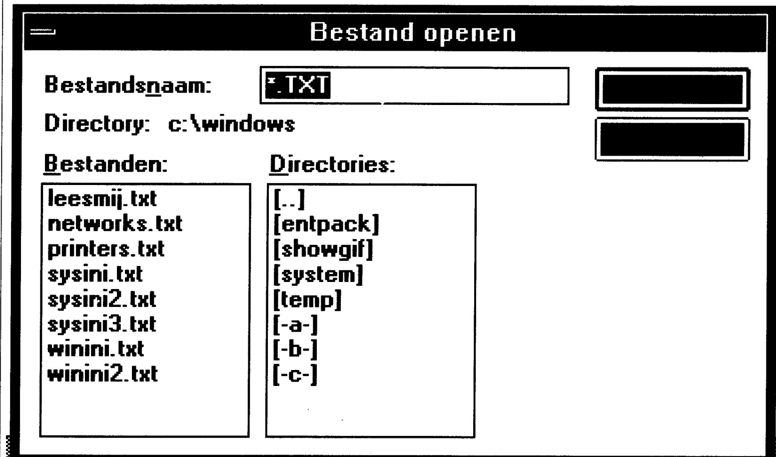
```
    LTEXT "&Name:", IDC_FILENAME, ...
    EDITTEXT IDC_EDIT, ...
    LISTBOX, IDC_LISTBOX, ...
    DEFPUSHBUTTON "&Open", IDOK, ...
    ...
END
```

Hierin zijn de afmetingen en het uiterlijk van de dialog box gedefinieerd en staan bovendien de gebruikte controls zoals LTEXT, EDITTEXT, LISTBOX en DEFPUSHBUTTON aangegeven. LTEXT geeft aan dat een bepaalde tekst links uitgelijnd moet worden weergegeven. EDITTEXT opent een klein venstertje waarin een bestandsspecificatie kan worden ingevoerd. LISTBOX opent een venster met de beschikbare bestanden, terwijl DEFPUSHBUTTON een default push button met de tekst 'Open' definieert. Het '&' teken staat vlak voor de letter waarmee de keuze ook gemaakt kan worden in combinatie met de Alttoets. In het voorbeeld staan puntjes waar normaliter de afmetingen en lokaties van de controls zouden staan.

Een voordeel van dialog boxes is dat het Windows-systeem automatisch een groot deel van de interactie tussen het programma en de gebruiker voor zijn rekening neemt. Het ontwerpen van een dialog box kan handmatig gebeuren in een conventionele editor. Eenvoudiger is het echter om gebruik te maken van een speciale editor; een van de vele tools die bij de Microsoft Windows SDK (Software Development Kit) meegeleverd wordt.

In een dialog editor kunnen de controls met behulp van de muis eenvoudig naar de gewenste positie van het venster worden gesleept zonder dat de exacte coördinaten worden aangegeven. De toolkit zal dan zelf een bestand aanmaken met een definitie zoals boven gegeven, inclusief de juiste coördinaten. Een ander voordeel van een dialog editor is dat het resultaat al tijdens de ontwerpfase te zien is. Iedere dialog box heeft een corresponderende dialog-functie met een eigen message-afhandeling waar onder andere de berichten van de controls terecht komen. Zo'n dialog-functie ziet er vaak als volgt uit:

```
BOOL FAR PASCAL OpenDlg( HWND hDlg,
                         WORD message,
                         WORD wParam,
                         LONG lParam )
{
```

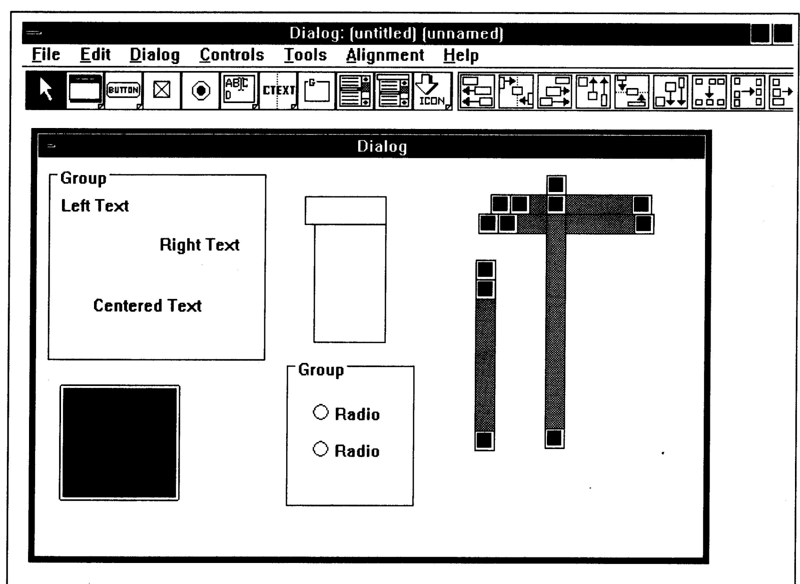


File Open Dialog

```
switch ( message )
{
  case WM_COMMAND:
    switch ( wParam )
    {
      case IDOK:
        ... Open bestand ...
      case IDCANCEL:
        EndDialog( hDlg, FALSE );
        return TRUE;

      case IDC_LISTBOX:
        ... Update listbox ...
    }
    break;
  }
}
```

Het Windows-systeem zal bij het activeren van de Open-button via de main message loop een IDOK-bericht naar de OpenDlg-functie sturen. Op vergelijkbare wijze wor-



Een Dialog Editor met diverse controls

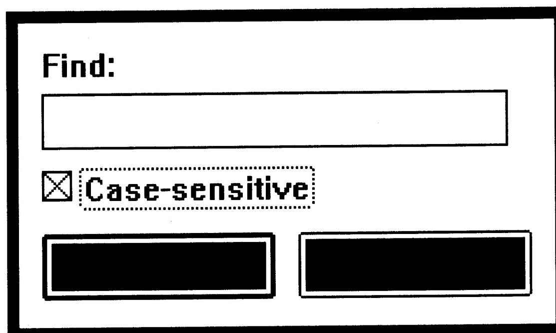
den berichten van andere controls, zoals de LISTBOX, verzonden aan de dialog-functie.

Nu het uiterlijk van de dialog box en de dialog-functie gedefinieerd zijn, kunnen we gaan kijken hoe het hoofdprogramma een dialog box aanmaakt.

```
  /* Start dialoog */
lpOpenDlg = MakeProcInstance( OpenDlg,
                              hInst );
DialogBox( hInst, "Open",
           hWnd, lpOpenDlg );
FreeProcInstance( lpOpenDlg );
```

Deze code wordt aangeroepen zodra de Open-optie uit het File-menu gekozen wordt. De MakeProcInstance-functie geeft een adres voor de OpenDlg-functie terug, die tijdelijk in lpOpenDlg (van het type FARPROC) wordt opgeslagen. Vervolgens wordt de dialog box geopend met de DialogBox-functie, en de controle gaat over naar de dialog box. Als de gebruiker het dialoog beëindigt, wordt lpOpenDlg met FreeProcInstance weer vrijgegeven. Dit proces wordt telkens na selectie van de Open-optie herhaald.

Hieruit wordt het belangrijkste kenmerk van een Modal dialog box duidelijk: de controle wordt na het openen van de dialoog volledig overgedragen aan de dialog-functie. De meest dialog boxes zijn Modal, zoals de optie Pagina Layout uit het Document-menu van Windows Write. Dit in tegenstelling tot de Modeless dialog box, die we nu zullen behandelen.



Een Find Dialog

Het aanmaken van een Modeless dialog box

Stel dat we een Find dialog box hebben om in bestanden op schijf naar een bepaalde tekenreeks te zoeken. We zouden het immers handig vinden als het venster zichtbaar blijft. Een Modeless dialog wordt in het .RC-bestand anders gedefinieerd dan een Modal box, en zullen meestal een WM_POPUP-stijl krijgen.

Het grootste verschil tussen een Modeless en een Modal dialog box is echter de manier van creatie en afsluiten. Een Modeless dialog box wordt namelijk gecreëerd met de CreateDialog-functie in plaats van de DialogBox-functie. Bovendien wordt de dialoog afgesloten door het aanroepen van de functie DestroyWindow in de dialog-functie.

```
lpFindProc = MakeProcInstance( FindDlg,
                               hInst );
CreateDialog( hInst, "Find",
              hWnd, lpFindDlg );
```

In de dialog-functie FindDlg die verder vergelijkbaar is met OpenDlg, wordt het sluiten van de dialoog als volgt behandeld:

```
BOOL FAR PASCAL OpenDlg( HWND hDlg,
                         WORD message,
                         WORD wParam,
                         LONG lParam )
{
  switch ( message )
  {
    case WM_COMMAND:
      switch ( wParam )
      {
        ... andere mogelijkheden ...
        case IDCANCEL:
          DestroyWindow( hDlg );
          return TRUE;
      }
      break;
  }
}
```

Een voordeel van Modeless dialog boxes is tevens dat er meer tegelijk kunnen zijn geopend, bijvoorbeeld om twee zoek-acties tegelijk bij te houden. De optie 'Zoeken naar' uit het Zoeken-menu van Windows Write is een voorbeeld van een Modeless menu. Als dit menu geopend is kan men tussendoor naar het hoofdprogramma terugspringen door het edit-venster aan te klikken.

Controls

Controls worden in veel dialog boxes gebruikt, maar kunnen uiteraard ook in reguliere vensters gebruikt worden. Een control is een speciaal venstertje dat gebruikt kan wor-

Terminalmodi
☒ Regelterugloop
☐ Lokale echo
☒ Geluid

Kolommen
◉ 80 ○ 132

check boxen en radio buttons

den om een bepaald soort in- of uitvoer te verzorgen. Controls worden bestuurd door een control window functie, die door Windows al is gedefinieerd. Dit betekent in de praktijk dat programma's gebruik kunnen maken van een edit-window waarvoor slechts het definiëren van een edit-control noodzakelijk is, zoals in het Open-dialog geschiedde.

Windows biedt o.a. de volgende ingebouwde controls.

- Push-buttons, de bekende drukknoppen van Windows die vanaf versie 3.0 een drie-dimensionaal uiterlijk hebben. Een push-button kan een tekst bevatten of een afbeelding die meer informatie geeft over de aard van de button. De control-style die bij een push-button hoort, is BS_PUSHBUTTON; BS_DEFPUSHBUTTON kan gebruikt worden om de default button aan te geven.

- Radio-buttons worden meestal in groepen geplaatst en geven de gebruiker de mogelijkheid precies één keuze te selecteren. Een radio-button wordt gekenmerkt door een kleine cirkel voor de tekst van de keuze. Indien een radio-button wordt geselecteerd, worden alle andere in dezelfde groep automatisch gedeselecteerd en wordt de cirkel van de gekozen optie gevuld met een massief zwart rondje. De stijl van een radio-button heet BS_RADIOBUTTON.

- Check boxes zijn aankruisvakken die zowel losstaand als in een groep gebruikt kunnen worden, en die individueel gekozen kunnen worden. Check boxes worden gekenmerkt door een vierkantje, waarin een kruisje verschijnt indien de optie momenteel geactiveerd staat. BS_CHECKBOX is de stijl die men voor check boxes moet gebruiken. Door 'Terminal Voorkeursinstellingen' te kiezen uit het Instellingen-menu van het Windows Terminal-programma, kan men zien wat het verschil is tussen radio-buttons en check boxes.

- Edit controls kunnen in diverse vormen gebruikt worden; zowel enkele regels als meer regels kunnen ge-edit worden. Het laatste type kan dus fungeren als een kleine tekst-editor zoals NotePad. Het Open-dialoogkader dat we al eerder hebben behandeld, bevat een edit control om de bestandsspecificatie te kunnen invoeren. Edit controls kunnen met de stijlen ES_LEFT en ES_MULTILINE gecreëerd worden.

- Static controls zijn vensters die gebruikt kunnen worden om er tekst in weer te geven. Tekst kan zowel links en rechts uitgelijnd worden, door respectievelijk SS_LEFT en SS_RIGHT als stijl te gebruiken.

- List boxes zijn vensters die een lijst tekststrings bevatten, waaruit een keuze kan worden gemaakt. Indien het venster te klein is om alle elementen te bevatten, kan de gebruiker met behulp van een scroll-bar door de lijst bladeren en een item selecteren. Een voorbeeld van een list box is het bestandsvenster van het Open-dialoogkader. De inhoud van een list box is niet statisch, maar kan gewijzigd worden, bijvoorbeeld omdat de gebruiker een andere bestandsspecificatie

aangeeft. Een voorbeeld van list boxes kan gevonden worden in de optie 'Lettertypen' van het Windows Configuratiescherm. Met LBS_STANDARD wordt de stijl aangegeven van de standaard list box. Er bestaan speciale list boxes waarmee meer elementen in een keer geselecteerd kunnen worden, of die uit meer kolommen bestaan.

° Een combo box bevat ook een lijst van keuzen. Er zijn drie typen combo boxes: simple, drop down en drop down list combo boxes. Een simple combo box heeft een edit control als bovenste element, dat altijd getoond wordt. Daaronder kan in een list box een aantal elementen worden 'gehangen'. Indien de gebruiker een woord intypt dat in deze lijst voorkomt, wordt deze automatisch als bovenste element in de list box gezet. Bij een drop down combo box wordt deze lijst alleen getoond als de gebruiker het drop down symbool aanklikt. Bij een drop down list combo box kan de gebruiker niet intypen wat hij wil, maar moet een keuze worden gemaakt uit de items uit de onderliggende list box. De optie 'Internationaal' van het Configuratiescherm van Windows bevat diverse drop down combo boxes. Afgezien van de multikolom mogelijkheid kunnen aan combo boxes vele andere eigenschappen worden toegekend, zoals het sorteren van de elementen.

Conclusie

Dialog boxes vormen een onmisbaar onderdeel van Windows-applicaties. De vele controls die een dialog box kan bevatten maken dit type venster een zeer veelzijdig hulpmiddel voor vele doeleinden.

Do What I Mean utility

Het TSR-programma dat in dit artikel besproken wordt, vult een onvolledig ingevoerde bestandsnaam op de command-line aan tot een bestaande naam uit de huidige directory.

```
[C:\]dwim
DWIM version 1.0
Hotkey = Alt-Z

[C:\]type info.txt
De naam van de directory is \Usr\Lalonde\SigPlan\Vol24Nr4\WorkShop\Notes\PS

We typen achtereenvolgens in:

CD \U <Alt-Z>           \usr

   \La <Alt-Z>          \usr\lalonde

   \Sig <Alt-Z>         \usr\lalonde\sigplan

   \vol24 <Alt-Z>       \usr\lalonde\sigplan\vol24nr4
   \w <Alt-Z>           \usr\lalonde\sigplan\vol24nr4\workshop
   \no <Alt-Z>          \usr\lalonde\sigplan\vol24nr4\workshop\notes
   \ps <Return>         \usr\lalonde\sigplan\vol24nr4\workshop\notes\ps

[C:\]cd \usr\lalonde\sigplan\vol24nr4\workshop\notes\ps

[C:\USR\LALONDE\SIGPLAN\VOL24NR4\WORKSHOP\NOTES\PS]
```

Het programma is resident, dus plaatst zichzelf permanent in het geheugen, waarna het met een speciale toetscombinatie kan worden opgeroepen. Deze toetscombinatie is in het huidige programma Alt-Z. Het programma kijkt naar het laatst ingetypte woord, en probeert een bestands- of directorynaam te vinden die daarop lijkt. Wordt zo'n naam gevonden, dan wordt het ingetypte woord aangevuld. Zo kan met 'TYPE Y' plus Alt-Z eenvoudig het bestand YR91RVNS.TXT bekeken worden.

De werking is vrij eenvoudig. Ik heb voor de implementatie van het hulpprogramma de Zortech C/C++ compiler gekozen, omdat deze TSR-mogelijkheden biedt in de standaard-library. De twee variabelen TSR_HOTSHIFT en TSR_HOTSCAN definiëren de de hot-key die in dit geval dus Alt-Z is. Met tsr_install() wordt het programma resident gemaakt. Indien DWIM voor de tweede keer wordt uitgevoerd van de commandline, zal het programma zichzelf uit het geheugen verwijderen met behulp van tsr_uninstall(). Vermeldenswaardig is verder dat de KbdStuff-routine een string in de toetsenbord-buffer stopt, na onder andere wat gemanipuleer met far pointers.

Met DWIM (Do What I Mean) kan eenvoudig vanaf de root naar bijvoorbeeld de directory \Document\Artikel\DWIM gesprongen worden. Hiervoor typt men CD \D in, gevolgd door Alt-Z. Het DWIM-programma zal vervolgens de naam Document aanvullen, waarna '\' ingetypt wordt gevolgd door de letter 'A'. Nogmaals Alt-Z levert op dat de naam 'Artikel' wordt gevonden, waarna return kan worden ingegeven, enzovoort. De huidige versie van DWIM vult alleen bestandsnamen aan die als eerste in de directory-listing gevonden worden. Komt in dat geval niet de gewenste naam te voorschijn, dan kan gezocht worden met twee letters, enzovoort. Indien slechts één bestand aanwezig is in de huidige directory, geeft DWIM

de naam van dat bestand ook als er geen beginletters zijn ingetypt.

Het programma neemt na installatie ongeveer 20 K aan geheugenruimte in beslag, maar kan hoog geladen worden, bijvoorbeeld met LoadHi van QuarterDeck. In dat geval neemt DWIM geen conventioneel geheugen in beslag. Het DWIM-programma zou op diverse manieren uitgebreid kunnen worden. Omzetten naar een andere compiler is mogelijk door gebruik te maken van een TSR-toolkit, zoals die in het public-domain circuit beschikbaar zijn.

```
/* DWIM.C
 * Do What I Mean TSR-programma voor hulp
 * bij onvolledige bestandsnamen
 * Compiler: Zortech C/C++
 * Gecompileerd met versie 2.1 */

#include <disp.h>
#include <tsr.h>


  /* Hot-key: Alt-Z */
int TSR_HOTSCAN = SCAN_Z;
int TSR_HOTSHIFT = ALT;
  /* Handtekening */
char tsr_fprint[20] = "Do What I Mean";
extern int _okbigbuf = 0;

int Debug = 0;
int Mode = POPONLY;
struct FIND *f;

  /* Prototypes: */
unsigned char WhereX(), WhereY(),
              GetCharAt(int x,int y);
void KbdStuff(char*);
int IsValid(char), UniqueMask(char*);
int FindStrPos(char*,char*);
char *FindFirst(char*), *FindNext();

#define IdMsg     "DWIM version 1.0"
#define HotKeyMsg "Hotkey = Alt-Z"

  /* Hoofdprogramma: */
int main( int ArgC, char **ArgV )
{
  int i;

  puts( IdMsg );
  puts( HotKeyMsg );

  i = tsr_install( Mode );

  if ( i==1 )
  {
    i = tsr_uninstall();
    puts("\rDWIM uninstalled");
  }
  else
    puts("Error");

}

  /* Hoofdprogramma TSR: */
void popmain( void )
{
  char Typed[128], searchStr[128];
  int i, j;
  unsigned char c, x, y;
  char *FileName;

  disp_open();

    /* Vorige positie op het scherm */
  x = WhereX()-1;
  y = WhereY();

    /* Controleer of huidige positie leeg is.
     * Als dit niet zo is, doen we verder niets
     */

  if ( GetCharAt( x+1, y ) != ' ' )
    return;

    /* Indien vorige positie leeg, en er is maar
     * een entry in de directory, dan wordt die
     * bestandsnaam in de buffer gezet.
     */
  if ( GetCharAt( x, y ) == ' ' )
  {
    char *FileName = FindFirst( "*.*" );

    if ( FileName )
    {
        /* Er zijn bestanden */
      char *SecondFile = FindNext( );
      if ( SecondFile == 0 )
      {
        /* Slechts een bestand - OK */
        strlwr( FileName );
        KbdStuff( FileName );
      }
    }
    return;
  }

  /* Incomplete bestandsnaam aanvullen */
    /* Haal nu het ingetypte woord op */
  for ( i=x, j=0; i>0 &&
        IsValid(c=GetCharAt(i,y));
        i-- )
    Typed[j++] = c;
  Typed[j] = 0;

    /* Controleer of de naam uit te breiden is */
  if ( !j || Typed[0]=='\\' || Typed[0]==':' )
    return 0;

    /* Keer de inhoud om */
  strrev( Typed );

    /* Probeer een passende bestandsnaam
       te vinden */
  strcpy( searchStr, Typed );
  if ( strchr( searchStr, '.' ) )
    strcat( searchStr, "*" );
  else
    strcat( searchStr, "*.*" );

  FileName = FindFirst( searchStr );

    /* Indien gevonden */
  if ( FileName )
  {
    int Idx;

      /* Vanaf waar aanvullen? */
    Idx = FindStrPos( Typed, FileName );

      /* Plaats aanvullende tekens in de
         buffer van het toetsenbord */
    strlwr( FileName );
    KbdStuff( FileName+Idx );
  }
  else
    sound_click();
}

  /* Huidige schermpagina: */
unsigned char Page()
{
  union REGS r;
```

```
  r.h.ah = 15;
  int86( 0x10, &r, &r );
  return r.h.bh;
}

  /* Huidige kolom cursorpositie: */
unsigned char WhereX()
{
  union REGS r;
  r.h.ah = 3;
  r.h.bh = Page();
  int86( 0x10, &r, &r );
  return r.h.dl;
}

  /* Huidige rij cursorpositie: */
unsigned char WhereY()
{
  union REGS r;
  r.h.ah = 3;
  r.h.bh = Page();
  int86( 0x10, &r, &r );
  return r.h.dh;
}

  /* Plaats string in KBD buffer: */
void KbdStuff( char *Str )
{
  int i, j;
  typedef int far *fip;
  fip Start, End, KbdBuffer;

  if ( (i=strlen(Str))  16 )
    i = 16;

  Start = (fip)MK_FP(0x40,0x1A);
  End   = (fip)MK_FP(0x40,0x1C);
  KbdBuffer =(fip)MK_FP(0x40,0x1E);

  *Start = 0x1E;
  *End   = 0x1E + 2*i;

  for ( j=0; j<i; j++ )
    KbdBuffer[j] = Str[j];
}

  /* Controleer op geldige tekens */
int IsValid( char c )
{
  return isalnum(c) ||
 strchr( "-_\\.:", c );
}

  /* FindStrPos
   * Vind de index in subStr vanaf waar
   * substitutie moet plaatsvinden.
   * FindStrPos("\AU","AUTOEXEC.BAT")
   *   geeft de waarde 2
   */
int FindStrPos( char *Str, char *subStr )
{
  int i = strlen( Str ) - 1,
      j = i;

     /* Indien string leeg, return 0 */
  if ( Str == 0 )
    return 0;

  while ( Str[i] != '\\' &&
  Str[i] != ':' && i> 0 )
    i—;

  if ( i==0 && Str[i] != '\\' && Str[i] != ':' )
    return j - i + 1;
  else
    return j - i;
}

  /* FindFirst
   * Zoek naar eerst voorkomende bestandsnaam
   * Slaat '.' en ".." altijd over.
   */
char *FindFirst( char *Str )
{
  char *Result = 0;
  f = findfirst( Str, FA_DIREC );

  if ( *(f-name) == '.' )
    Result = FindNext( );
  else if ( f )
    Result = f-name;

  return Result;
}

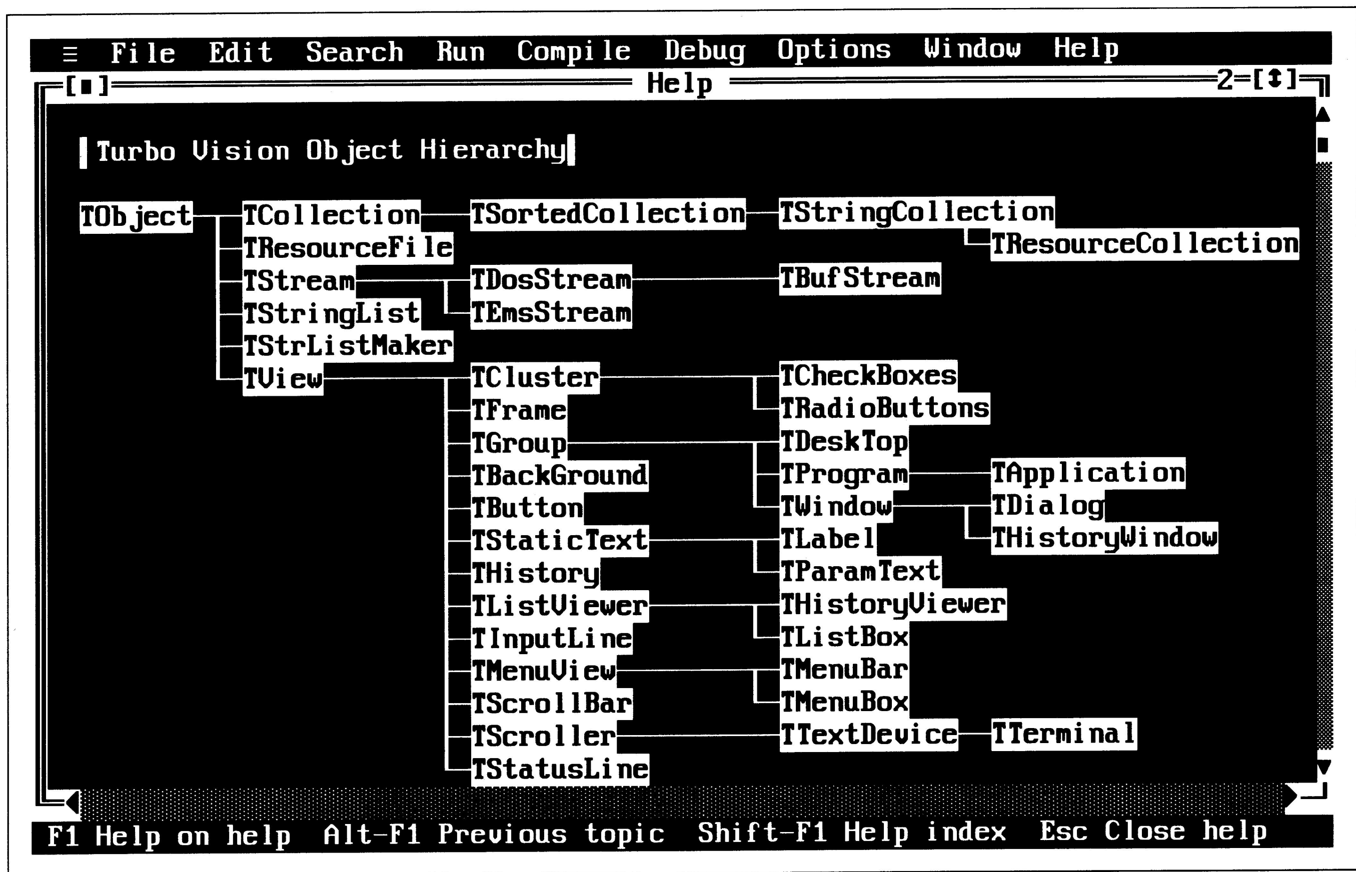
  /* FindNext
   * Zoek naar de eerstvolgende entry na
   * een FindFirst-aanroep
   */
char *FindNext( )
{
  char *Result = 0;

  f = findnext( );

  if ( *(f-name) == '.' )
    Result = FindNext( );
  else if ( f )
    Result = f-name;

  return Result;
}

  /* GetCharAt
   * Geef het teken op positie (x,y)
   * van het beeldscherm
   */
unsigned char GetCharAt( int x, int y )
{
  return disp_peekw( y, x ) & 0xFF;
}
```



TurboVision

Framework voor het schrijven van text-based programma's

De komst van Turbo Pascal 6.0 was voor vele Pascal-gebruikers een verademing wat betreft de gebruikersomgeving. Eindelijk konden dan ook zij genieten van resizable en verplaatsbare windows, muisondersteuning enzovoort. De 'look and feel' van de populairste Pascal compiler was eindelijk op een aanvaardbaar niveau gekomen. De metamorfose die de IDE had ondergaan was geheel te danken aan het bij Borland zelf ontwikkelde framework TurboVision.

TurboVision is een applicatie-framework, een geheel van objecten die het bij elkaar mogelijk maken een applicatie te voorzien van een aantrekkelijke gebruikersomgeving. De grootste voordelen van zo'n applicatie-framework zijn de afscherming van moeilijke implementatieprincipes en het hergebruik van code. Borland brengt nog een ander applicatie-framework uit, ObjectWindows, waarvan u ook in deze DOS-Special een beschrijving vindt.

OOP

Een applicatie-framework zoals TurboVision is geschreven in object-georiënteerde code. Wanneer u deze techniek nog niet zo meester bent raad ik u aan eerst het artikel uit de vorige DOS-Special over OOP in TP te lezen. Het gebruik van virtuele methoden dient u ook te begrijpen. In deze DOS-Special staat een artikel over het gebruik hiervan.

Event driven programming

De twee belangrijkste zaken die TurboVision u biedt zijn window-support en het afhandelen van events. Dit afhandelen van events gebeurt ook in uw oude interactieve programma's. Op basis van de input die de gebruiker geeft maakt u beslissingen over het verdere verloop van het programma. Wanneer een gebruiker bijvoorbeeld te kennen geeft dat het programma verlaten dient te worden, dan roept u een procedure aan die deze afhandeling verzorgt. In uw programma staat dan bijvoorbeeld een procedure 'GetInput', die er alsvolgt uit kan zien:

```
Procedure GetInput;

Var K : Char;
    Quit :  Boolean;

Begin
 Quit:=False;
 Repeat
  K:=ReadKey;
  Case K Of
    'C' : CopyFile;
    'D' : DeleteFile;
    'E' : EditFile;
    'M' : MoveFile;
    'Q' : Quit:= True;
    'V' : ViewFile;
  Until Quit;
End;
```

Zulke Case statements kunnen aardig complex en onoverzichtelijk worden. Wanneer aan een aantal voorwaarden voldaan moet zijn om een bepaalde procedure of functie aan te roepen is dit eerder het geval. Er zijn nog wel meer events te verzinnen dan alleen keyboard-invoer. Muishandelingen zijn namelijk ook events. Event driven programming neemt u deze, en nog meer, moeilijkheden uit handen.

Alle events worden afgevangen door TurboVision en naar de juiste bestemming in de applicatie gestuurd. Stel bijvoorbeeld dat u in Turbo Pascal twee windows geopend hebt waarvan de huidige het andere gedeeltelijk overdekt. Wanneer u de muiscursor nu op het onderliggende window plaatst en de linkerknop indrukt, dan wordt een event gestuurd naar het onderliggende window dat dit het huidige window wordt. Het window wordt nu dan ook in z'n geheel op het scherm gezet en niet meer overlapt door het andere window.

TApplication

Om te laten zien hoeveel werk het framework TurboVision ons bespaart zullen we een voorbeeld geven dat een van de meest wezenlijke onderdelen van een TurboVision-applicatie in zich heeft.

```
Program IntroInTV;

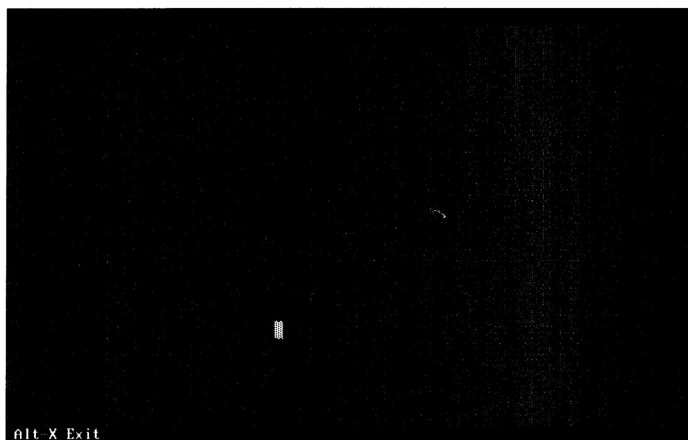
Uses App;

Type
  TMyApplication = Object(TApplication)
                   End;

Var
  MyApp: TMyApplication;

Begin
  MyApp.Init;
  MyApp.Run;
  MyApp.Done;
End.
```

Dit ziet er allemaal niet zo schrikbarend uit, maar wanneer u deze code compileert en runt zult u zien wat de kracht van zo'n framework als TurboVision is. Het resultaat is het volgende beeld.



Om dit zelf voor elkaar te krijgen had u toch een aantal regels code meer nodig gehad dan deze twaalf, zeker wanneer u ook nog met een muis zou moeten werken. Wat u voor u ziet is een desktop met linksonderin op de statusline een message 'Alt-X Exit'. Wanneer u deze toetscombinatie gebruikt springt u uit het programma. Het enige wat u hiervoor gedaan moet hebben is een object 'TMyApplication' afleiden van 'TApplication', daar een instantie 'MyApp' van creëren die geïnitialiseerd (Init), gerund (Run) en opgeruimd (Done) wordt. TApplication is een zeer belangrijk object in de TurboVision object-hiërarchie en wordt in elke TurboVision-appli-

catie gebruikt. U moet altijd een afgeleid object van TApplication creëren, die u zoals hier bijvoorbeeld TMyApplication kunt noemen. Het object TMyApplication voegt niets toe aan de methoden en datamembers die zij van TApplication erft, zodat de message 'Run' die 'MyApp' zendt de functiebody van 'Run' uit 'TApplication' zal activeren. In pseudocode ziet die body er alsvolgt uit:

```
Repeat
 Get an event;
 Handle the event;
Until Quit;
```

Het zal u niet verbazen dat dit erg lijkt op de code die we eerder als voorbeeld gaven. 'TApplication' is verantwoordelijk voor het afhandelen van de meeste events.

Het initialiseren van een instantie van TApplication geeft dus een desktop met een statusline en een menubar.

Views

Iets dat zichtbaar is op het scherm heet een view. Een view is altijd een object, in dit geval dus TApplication. Andere views zijn bijvoorbeeld dialog boxes en scrollbars. Een view kan een aantal subviews hebben, zoals in het geval van TApplication, waarna het een groep genoemd wordt. TApplication is de eigenaar van drie subviews, te weten de menubar, de statusline en de desktop. Dit heeft niets met een object-hiërarchie te maken, het wil alleen zeggen dat TApplication het beheer heeft over deze drie views. Als u naar de object-hiërarchie van TurboVision kijkt ziet u dat TStatusLine rechtstreeks van TView is afgeleid, TMenuBar via TMenuView en TDeskTop via TGroup is afgeleid van TView.

Alles wat afgeleid is van TView, rechtstreeks of indirect, is een view. Wanneer een instantie van zo'n object wordt geïnitialiseerd zal dan ook iets op het beeldscherm verschijnen. Een instantie van TView zelf laat niets op het beeldscherm verschijnen. Het krijgt alleen het bezit van een bepaald deel van het scherm, hetgeen altijd een rechthoek is. TView is in het bezit van een virtuele Drawmethode. Omdat deze zelf niets doet moet Draw overschreven worden in afgeleide objecten van TView. Merk op dat een subview ook weer een groep kan zijn.

Mute object

Objecten die niet zijn afgeleid van TView zijn mute objecten, hetgeen vrij vertaald 'stomme objecten' betekent. Ze zijn niet zichtbaar op het scherm zodat u daar niet kunt zien waar zij mee bezig zijn. Mute objecten worden gebruikt om bijvoorbeeld de communicatie met een randapparaat uit te voeren. Wanneer zij een melding willen maken van een resultaat van hun activiteiten moeten zij dat in een TV-applicatie altijd via een view doen.

Statusline

Nu is de statusline die TurboVision u verstrekt bij het initialiseren van een instantie van TApplication natuurlijk niet één die uitgebreid genoeg is. TV bezit gelukkig de mogelijkheid om de statusline uit te breiden met een aantal andere items. Wanneer we bijvoorbeeld windows zouden willen kunnen sluiten, dan kunnen we dat via de statusline weergeven. We overschrijven de methode InitStatusLine uit TApplication en initialiseren een nieuwe statusline.

```
Program ShowStatusLine;

Uses Objects, Drivers, Views, Menus, App;

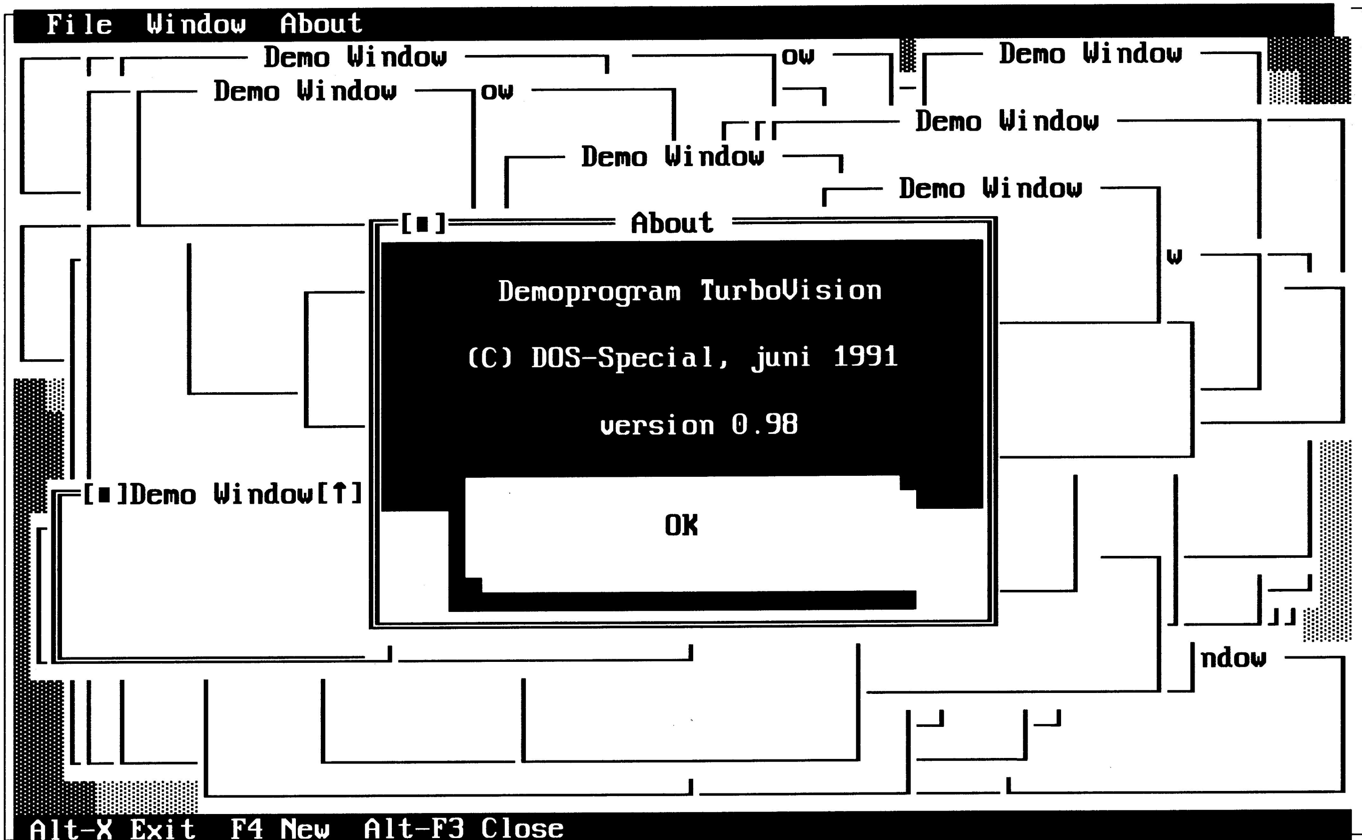
Const cmNewWin = 200;

Type
  TMyApplication = Object(TApplication)
     Procedure InitStatusLine; virtual;
                End;

Procedure TMyApplication.InitStatusLine;
Var R: TRect;
Begin
  GetExtent(R);
  R.A.Y := R.B.Y - 1;
  StatusLine := New(PStatusLine, Init(R,
    NewStatusDef(0, $FFFF,
      NewStatusKey('~Alt-X~Exit',kbAltX,
                  cmQuit,
      NewStatusKey('~F4~ New',kbF4,cmNew,
      NewStatusKey('~Alt-F3~ Close',
                  kbAltF3, cmClose,
      Nil)),
    Nil)
  ));
End;

Var
  MyApplication : TMyApplication;

begin
  MyApplication.Init;
```




```
  MyApplication.Run;
  MyApplication.Done;
End.
```

Let erop dat de overschreven methode InitStatusLine virtueel moet zijn, omdat die dat in TApplication ook al was. De variabele R is nodig om de schermgrootte mee op te halen. De statusline wordt dan een regel boven de onderste regel geplaatst. Met het statement 'StatusLine:=...' wordt de eigenlijke statusline gecreëerd. Deze statusline zal twee items hebben, namelijk 'ALT-X Exit' en 'ALT-F3 Close'. Wanneer met de linkermuisknop op de string 'ALT-X Exit' gedrukt wordt zal het standaard-commando cmQuit geactiveerd worden. Ook het event 'kbAltX' zal dit veroorzaken, dus wanneer gelijktijdig Alt en X worden ingedrukt. Bij het aangeven van de string die op de statusline komt worden de characters die tussen tildes staan (~) met een grotere intensiteit weergegeven. De twee commando's cmQuit en cmClose zijn standaard aanwezig in TurboVision, maar we kunnen ook zelf commando's toevoegen. In dit geval hebben we een nieuw commando cmNewWin in de declaratie van constanten toegevoegd, die we in de statusbar gebonden hebben aan de toets F4 en de het statusline item 'F4 New'.

Menubar

Om al deze mogelijkheden ook vanaf de menubar te kunnen kiezen is het noodzakelijk om ook de methode InitMenuBar uit TApplication te overschrijven. Het aanmaken van zo'n nieuwe menubar gaat ongeveer hetzelfde in z'n werk als het aanmaken van een nieuwe statusbar:

```
Const cmFileOpen = 201

Procedure TMyApplication.InitMenuBar;
Var R: TRect;
Begin
  GetExtent(R);
  R.B.Y := R.A.Y + 1;
  MenuBar := New(PMenuBar, Init(R,
   NewMenu(
    NewSubMenu('~F~ile', hcNoContext,
     NewMenu(
      NewItem('~O~pen', 'F3', kbF3,
              cmFileOpen, hcNoContext,
      NewItem('~N~ew', 'F4', kbF4,
              cmNewWin, hcNoContext,
      NewLine(
      NewItem('E~x~it', 'Alt-X', kbAltX,
              cmQuit, hcNoContext,
      Nil))))),
```

```
    NewSubMenu('~W~indow', hcNoContext,
     NewMenu(
      NewItem('~N~ext', 'F6', kbF6,
             cmNext, hcNoContext,
      NewItem('~Z~oom', 'F5', kbF5,
             cmZoom, hcNoContext,
      Nil))),
    Nil))
  )));
End;
```

We hebben weer een nieuw commando cmFileOpen geïnitialiseerd. Ook hier wordt eerst de grootte van het scherm opgehaald, waarna de menubar op de bovenste regel wordt geplaatst. Ook hier geldt dat de tekst tussen twee tildes highlighted op het beeldscherm komt te staan. Met NewSubMenu wordt allereerst een nieuw pulldown-menu 'File' aangemaakt. Deze heeft zelf aan menu's 'Open', 'New' en 'Exit'. 'NewLine' geeft aan dat er tussen de items 'New' en 'Exit' een streep komt te staan. Een ander submenu is 'Window'. Deze heeft als menu's 'Next' en 'Zoom'. Hier zijn weer twee standaard commando's van TurboVision gebruikt, cmNext en cmZoom. Het commando cmNext zorgt ervoor dat het volgende window het actuele window wordt. Het commando cmZoom geeft aan dat het actuele window het totale beeldscherm, behalve de menubar en de statusline tot zijn beschikking krijgt. Ook hier zijn alle commando's weer gebonden aan een aantal events. Het commando cmNext wordt geactiveerd door de toets F5 en natuurlijk door met de muis dit item aan te klikken. U dient dus in de declaratie van TMyApplication deze methode aan te geven en weer virtueel te maken. Door het polymorfistische gedrag van TurboVision zal automatisch deze InitMenuBar worden aangeroepen (in plaats van die uit TApplication). Verbaas u niet over 'hcNoContext'. TurboVision maakt het u makkelijk om een context-gevoelige 'help' in te bouwen. Elke view kan dan een 'help' contextnummer krijgen, zodat het zoeken gemakkelijker wordt. De context 'hcNoContext' is een speciaal geval dat de huidige context niet verandert. Op deze helpfuncties zullen we hier niet verder ingaan.

Windows

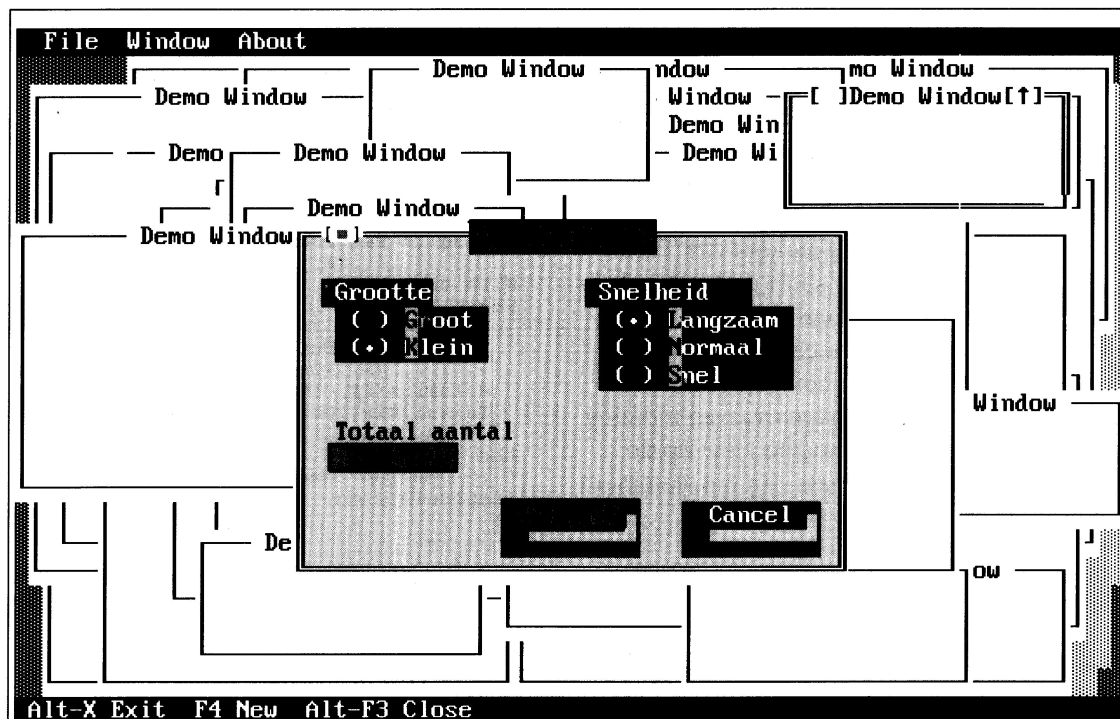
Om buiten de door TApplication gegeven statusline, desktop en menubar andere views op het scherm te krijgen moeten we weer een of andere afgeleide van TView instantiëren. Om een window op het scherm te kunnen krijgen moeten we een object afleiden van TWindow. Een TurboVision window is weer een zeer 'intelligent' object. Het weet hoe het zichzelf moet vergroten, verkleinen, sluiten en verplaatsen. Om een window te kunnen openen moeten we in TMyApplication een nieuwe methode aanmaken die een nieuw window kan openen. Omdat dit window pas geopend wordt wanneer een bepaald event plaatsvindt dat nog niet bekend is bij TV moet ook de methode HandleEvent uit TApplication overschreven worden. Deze was virtueel dus wordt dat nu weer. In deze nieuwe HandleEvent moeten we altijd de expliciete HandleEvent uit TApplication aanroepen door middel van het statement TApplication.HandleEvent(Event);. Wanneer nu het commando cmNewWin wordt gegeven, dan wordt de methode NewWindow uit MyApplication uitgevoerd. Wanneer het commando een andere was dan cmNewWin wordt uit de methode gesprongen. Een nieuwe constante WinCount wordt geïnitialiseerd die bijhoudt hoeveel windows er geopend zijn.

```
Const WinCount : Integer = 0;
      PDemoWindow = ^TDemoWindow;
      TDemoWindow = object(TWindow)
                    End;

Procedure TMyApp.HandleEvent
          (Var Event : TEvent);
Begin
  TApplication.HandleEvent(Event);
  If Event.What = evCommand then
  Begin
    Case Event.Command Of
      cmNewWin: NewWindow;
    Else
      Exit;
    end;
    ClearEvent(Event);
  End;
End;

Procedure TMyApp.NewWindow;
Var
  Window: PDemoWindow;
  R: TRect;
Begin
  Inc(WinCount);
  R.Assign(0, 0, 25, 6);
  R.Move(Random(60), Random(15));
  Window := New(PDemoWindow, Init
            (R,'RandomWindow',WinCount));
  DeskTop^.Insert(Window);
End;
```

In de methode NewWindow wordt allereerst de variabele WinCount met 1 verhoogd. De variabele R bevat de grootte en de plaats van



het te openen window, de plaats hier verkregen met random statements. Dan wordt het window geïnitialiseerd door de aanroep 'Window:=New(...)'. Het statement 'DeskTop^.Insert(Window)' is van groot belang bij het werken met TurboVision. Wat u hier doet is deze view op de groep 'DeskTop' plaatsen. U kunt alle views op een groep zoals DeskTop plaatsen, waarna de groep de eigenaar wordt van die view. Die groep heet dan de eigenaar, terwijl de view de subview heet. Een handige eigenschap van deze methode is dat wanneer de eigenaar gesloten wordt deze meteen ook de destructors aanroept van al zijn subviews.

Dialog boxes

De laatste soort views die we in dit artikel zullen bespreken zijn dialog boxes. Dialog boxes zijn, de naam zegt het al, bepaalde views die de communicatie tussen de gebruiker en het programma verzorgen. In een dialog box kan een gebruiker bijvoorbeeld aangeven welke file geopend moet worden of de preferences aangeven voor een bepaalde applicatie. TurboVision biedt u het object TDialog aan om een dialog box mee te maken. Anders dan bij bijvoorbeeld TApplication zal van TDialog niet een nieuw object worden afgeleid, aangezien twee dialog boxes er wel anders uitzien, maar niet in gedrag van elkaar verschillen (OOP!). We initialiseren weer een nieuw commando dat een bepaalde dialog box kenmerkt. In de menubar kunnen we dan een bepaald item toevoegen dat de dialog box op het scherm zou zetten. Een dialog box wordt alsvolgt gedefinieerd:

```
Uses Dialogs;

cmNewDialog        = 203;

Procedure TMyApplication.NewDialog;
Var
  Dialog: PDemoDialog;
  R: TRect;
Begin
  R.Assign(0, 0, 40, 13);
  R.Move(38, 10);
  Dialog := New(PDemoDialog, Init
            (R, 'Demo Dialog'));
  DeskTop^.Insert(Dialog);
End;
```

U zou in dit geval dus het Case statement van de eventhandler moeten uitbreiden met:

```
cmNewDialog : NewDialog;
```

en de methode NewDialog in het object TMyApplication moeten toevoegen. De enige verschillen met een window is dat een dialog box de grijze kleur heeft, niet te vergroten of te verkleinen is en geen nummer heeft. Een druk op de Esc-toets zorgt ervoor dat de dia-

log box weer verdwijnt. U ziet dat de dialog-box net als een window weer op de desktop geplaatst wordt. Dit is een voorbeeld van een non-modal dialog box, hetgeen wil zeggen dat de controle over de applicatie niet bij de dialog box ligt. Het is mogelijk dat een andere view weer actief wordt. Meestal zou u dat niet willen, zodat de makers van Turbo-Vision ook in een ander soort dialog box hebben voorzien, de zogenaamde modal dialog-box. U kunt nu niet naar een andere view gaan voordat de dialog box gesloten is. Het enige verschil in initialiseren van zo'n dialog box is het verschil van hoe de view op de desktop te zetten. In plaats van het statement

```
DeskTop^.Insert(Dialog);
```

schrijft u

```
Control:=DeskTop^.ExecView(Dialog);
```

Controls

Nu onderhoudt de dialog box die we tot nu toe hebben ontworpen weinig communicatie met de gebruiker. Er is geen enkele mogelijkheid tot interactie. TurboVision biedt u een aantal mogelijkheden waarop de gebruiker kan reageren. Een daarvan is het gebruik van buttons. Buttons worden in een dialog box geïnitialiseerd door een statement van de vorm:

```
R.Assign(x1,y1,x2,y2);
Dialog^.Insert(New(PButton,Init
                (R,'naam',command,flag)));
```

U ziet dat net als bijvoorbeeld een window in de desktop werd geplaatst hier een button in de dialog box geplaatst wordt. De variabele R van het type 'TRect' geeft weer de grootte van de rechthoek aan. Het commando kan in dit geval een van de vooraf gedefinieerde zijn van TurboVision, zoals cmOK, cmCancel, cmYes of cmNo. Let wel, de button die het commando cmOK activeert hoeft niet de naam 'OK' te cmNo, al programmeert dat cmNo wel een stuk gemakkelijker. De flag die in dit geval meegegeven moet worden geeft aan of de button de default button is of niet. De flags bmDefault en bmNormal zijn de te gebruiken flags hier. Om in een dialog-box een OK button en een Cancel button toe te voegen hebben we dus de volgende code nodig:

```
Procedure TMyApplication.NewDialog;
Var  Dialog: PDemoDialog;
     R: TRect;
     C: Word;
Begin
  R.Assign(20, 6, 60, 19);
  Dialog := New(PDemoDialog, Init
              (R, 'Demo Dialog'));
  With Dialog^ do
  Begin
    R.Assign(15, 10, 25, 12);
    Insert(New(PButton, Init
           (R,'~O~K', cmOK,bfDefault)));
    R.Assign(28, 10, 38, 12);
    Insert(New(PButton, Init
       (R,'Cancel',cmCancel, bfNormal)));
  End;
  C := DeskTop^.ExecView(Dialog);
  Dispose(Dialog, Done);
End;
```

Aangezien de gebruiker nu nog geen mogelijkheid heeft om keuzes weer te geven, er staan slechts twee buttons ter beschikking, zijn er in TurboVision mogelijkheden om invoer van de gebruiker te verkrijgen. Belangrijke daarvan zijn checkboxes en radiobuttons. Deze objecten, TCheckBoxes en TRadio-Buttons, zijn beide afgeleid van het 'abstracte' object TCluster en verschillen dan ook niet veel van elkaar. Zoals u in de figuur ziet kunt u me beide een aantal voorkeuren aangeven. Van een serie radiobuttons kan en moet er maar een gekozen worden, terwijl dit aantal bij checkboxes niet aan een limiet of ondergrens gebonden is. Om in onze dialogbox een aantal radiobuttons en check-boxes en radiobuttons toe te voegen hoeven we maar weinig te doen. Voor radiobuttons initialiseren we in de methode NewDialog een pointer B naar een view, die daarna als instantie van RadioButtons gaat fungeren. Merk op dat door het gebruik van polymorfisme alle virtuele methoden die in TView zijn gedefinieerd, het type van de pointer van B, niet worden uitgevoerd. Door het statement 'B:=New(PRadioButtons, Init(R,...))' wordt de body van de methoden die in het afgeleide object zijn gedeclareerd gebruikt, dus die in TCluster of TRadioButtons. Op elke plaats waar een base object verwacht wordt mag namelijk een afgeleid object verschijnen. Tijdens de executie van het programma wordt dan wel uitgezocht welke methode gebruikt moet worden. Voor de radiobuttons geldt hetzelfde. We verkrijgen dan

de volgende code voor het initialiseren van de dialogbox:

```
Procedure TMyApplication.NewDialog;
Var
  B: PView;
  Dialog: PDemoDialog;
  R: TRect;
  C: Word;

Begin
  R.Assign(20, 6, 60, 19);
  Dialog := New(PDemoDialog, Init
                (R,'Demo Dialog'));
  With Dialog^ do
  Begin
    R.Assign(3, 3, 18, 6);
    B:= New(PRadioButtons, Init(R,
      NewSItem('~V~ariabel',
      NewSItem('V~a~st',
      Nil)))
    ));
    Insert(B);
    R.Assign(2, 2, 10, 3);
    Insert(New(PLabel, Init(R,
              'Grootte', B)));
    R.Assign(22, 3, 34, 6);
    B:= New(PRadioButtons, Init(R,
      NewSItem('~L~angzaam',
      NewSItem('~N~ormaal',
      NewSItem('~S~nel',
      Nil)))
    ));
    Insert(B);
    R.Assign(21, 2, 33, 3);
    Insert(New (PLabel, Init(R,
                'Snelheid', B)));
    R.Assign(15, 10, 25, 12);
    Insert(New (PButton, Init(R,
       '~O~k', cmOKbfDefault)));
    R.Assign(28, 10, 38, 12);
    Insert(New (PButton, Init(R,
           'Cancel',cmCancel,bfNormal));
  End;
  C := DeskTop^.ExecView(Dialog);
  Dispose(Dialog, Done);
End;
```

Input line

Om de gebruiker via het toetsenbord data te laten invoeren hebben de makers van TurboVision een object TInputLine toegevoegd. Het programmeren van zo'n inputline is tamelijk complex, maar het gebruik van het object is zeer eenvoudig. Om de inputline in onze dialog box te plaatsen kunnen we weer de pointer B gebruiken die ook voor de radiobuttons is gebruikt. We moeten dan weer B als een nieuw type initialiseren, door hem als een pointer naar een InputLine te laten wijzen. Dit gebeurt met de volgende opdrachten:

```
R.Assign(3, 8, 37, 9);
B:=New(PInputLine, Init(R, 128));
Insert(B);
R.Assign(2, 7, 24, 8);
Insert(New(PLabel,Init(R,
                'Totaalaantal',B)));
```

Weer allemaal vrij simpel. Deze code komt in de methode NewDialog te staan, bijvoorbeeld na de initialisatie van de CheckButtons, maar natuurlijk voordat de dialog box getekend wordt.

Andere mogelijkheden

Buiten de hier besproken mogelijkheden om een dialog box op te vullen zijn er nog andere objecten. Het gebruik hiervan is hetzelfde als met bijvoorbeeld de inputline. Het object TStaticText schrijft een bepaalde string in de rechthoek die het ter beschikking krijgt. Een object van het type TListViewer biedt de gebruiker een één- of tweedimensionale lijst aan, waaruit de gebruiker items kan kiezen.

Het gebruik van scrollbars, die TurboVision ook op andere plekken aan de programmeur kan bieden, wordt ook in dit object ondersteund. TListBox is een object dat afgeleid is van TListViewer. Het beheert een TCollection van pointers naar strings. Een voorbeeld van een listbox is het file-selecteermenu in de IDE van Turbo Pascal 6.0. Ten slotte, een instantie van het type THistory geeft de gebruiker de mogelijkheid om eerder gegeven opties terug te halen uit een listbox.

Voorbeeldprogramma

Het voorbeeldprogramma bestaat uit alle onderwerpen die in dit artikel besproken zijn. Er zijn twee dialogboxes in verwerkt, waarvan één een aboutbox is en één informatie van de gebruiker opvraagt. In deze laatste, die met F2 is op te roepen, worden radiobuttons en een inputline gebruikt. De ingevoerde data wordt in de methode AnalyseDialogInput geanalyseerd. Het programma is buiten het gebruik van TurboVision een voorbeeld van hoe efficiënt een TV-applicatie met geheugen omspringt. U kunt met een gerust hart meer dan duizend windows openen, door dit in de dialogbox aan te geven.

```
Program TurboVisionDemo;

uses Crt, Objects, Drivers, Views, Menus, App,
Dialogs;

Const
  WinCount: Integer =   0;
  cmFileOpen        = 100;
  cmNewWin          = 101;
  cmNewDialog       = 102;
  cmAbout           = 103;
  WSmall            = 21;
  HSmall            = 5;
  WLarge            = 30;
  HLarge            = 10;


Type
  DialogData = Record
                CheckBox1,
                CheckBox2 : Word;
                InputLine : String[128];
               End;

  TMyApplication = Object(TApplication)
    MyDialogData        : DialogData;
    W,H,AdjX            : Integer;

    Constructor Init;

    Procedure HandleEvent(var Event: TEvent);
     Virtual;
    Procedure InitMenuBar;
     Virtual;
    Procedure InitStatusLine;
     Virtual;
    Procedure NewDialog;
    Procedure NewWindow;
    Procedure AnalyseDialogInput;
    Procedure About;
  End;

  PDemoWindow = ^TDemoWindow;
  TDemoWindow = object(TWindow)
               End;

  PDemoDialog = ^TDemoDialog;
  TDemoDialog = object(TDialog)
               End;


Constructor TMyApplication.Init;

Var R : TRect;

Begin
 TApplication.Init;
 With MyDialogData Do
 Begin
  CheckBox1:=0;
  CheckBox2:=2;
  InputLine:='10';
 End;
 W:=WLarge;
 H:=HLarge;
 Randomize;
 GetExtent(R);
 AdjX:=R.B.Y-25;
End;

Procedure TMyApplication.HandleEvent
           (var Event: TEvent);
Begin
  TApplication.HandleEvent(Event);
  If Event.What = evCommand
  Then
  Begin
    Case Event.Command Of
      cmNewWin: NewWindow;
      cmNewDialog: NewDialog;
      cmAbout     : About;
    Else
      Exit;
    End;
    ClearEvent(Event);
  End;
End;

Procedure TMyApplication.InitMenuBar;

Var R: TRect;

Begin
  GetExtent(R);
  R.B.Y := R.A.Y + 1;
  MenuBar := New(PMenuBar, Init(R, NewMenu(
    NewSubMenu('~F~ile', hcNoContext, NewMenu(
      NewItem('~O~pen', 'F3', kbF3, cmFileOpen,
hcNoContext,
      NewItem('~N~ew', 'F4', kbF4, cmNewWin,
hcNoContext,
      NewLine(
      NewItem('E~x~it', 'Alt-X', kbAltX, cmQuit,
hcNoContext,
      Nil))))),
    NewSubMenu('~W~indow', hcNoContext, NewMenu(
      NewItem('~N~ext', 'F6', kbF6, cmNext,
hcNoContext,
      NewItem('~Z~oom', 'F5', kbF5, cmZoom,
hcNoContext,
      NewItem('~D~ialog', 'F2', kbF2,
cmNewDialog, hcNoContext,
    Nil)))),

NewItem('~A~bout','F1',kbF1,cmAbout,hcNoContext,
    Nil))))
  ));
End;

Procedure TMyApplication.InitStatusLine;

Var R: TRect;

Begin
  GetExtent(R);
  R.A.Y := R.B.Y - 1;
  StatusLine := New(PStatusLine, Init(R,
    NewStatusDef(0, $FFFF,
      NewStatusKey('', kbF10, cmMenu,
      NewStatusKey('~Alt-X~ Exit', kbAltX,
cmQuit,
      NewStatusKey('~F4~ New', kbF4, cmNewWin,
      NewStatusKey('~Alt-F3~ Close', kbAltF3,
cmClose,
      Nil)))),
    Nil)
  ));
End;


procedure TMyApplication.NewWindow;

Var
  Window: PDemoWindow;
  R: TRect;
Begin
  Inc(WinCount);
  R.Assign(0, 0, W, H);
  If MyDialogData.CheckBox1 = 0
  Then
    R.Move(Random(51), Random(14+AdjX))
  Else
    R.Move(Random(60), Random(19+AdjX));
  Window := New(PDemoWindow, Init(
              R,'Demo Window', WinCount));
  DeskTop^.Insert(Window);
End;

Procedure TMyApplication.NewDialog;
```

```
Var
  B: PView;
  Dialog: PDemoDialog;
  R: TRect;
  C: Word;

Begin
  R.Assign(20, 6, 60, 19);
  Dialog := New(PDemoDialog, Init
              (R, 'Demo Dialog'));
  With Dialog^ do
  Begin
    R.Assign(3, 3, 14, 5);
    B:= New(PRadioButtons, Init(R,
          NewSItem('~G~root',
          NewSItem('~K~lein',
         Nil))
    ));
    Insert(B);
    R.Assign(2, 2, 10, 3);
    Insert(New(PLabel, Init(R, 'Grootte', B)));
    R.Assign(22, 3, 36, 6);
    B:= New(PRadioButtons, Init(R,
          NewSItem('~L~angzaam',
          NewSItem('~N~ormaal',
          NewSItem('~S~nel',
         Nil)))
    ));
    Insert(B);
    R.Assign(21, 2, 33, 3);
    Insert(New (PLabel, Init(R, 'Snelheid', B)));
    R.Assign(15, 10, 25, 12);
    Insert(New (PButton, Init(R,
              '~O~k', cmOK, bfDefault)));
    R.Assign(28, 10, 38, 12);
    Insert(New (PButton, Init(R,
              'Cancel', cmCancel, bfNormal)));
    R.Assign(3, 8, 37, 9);
    B:=New(PInputLine, Init(R, 128));
    Insert(B);
    R.Assign(2, 7, 24, 8);
    Insert(New(PLabel, Init(R,
             'Totaal aantal',B)));
  End;
  Dialog^.Setdata(MyDialogData);
  C := DeskTop^.ExecView(Dialog);
  If C <> cmCancel
  Then
   Begin
    Dialog^.GetData(MyDialogData);
    AnalyseDialogInput;
   End;
  Dispose(Dialog, Done);
End;

Procedure TMyApplication.AnalyseDialogInput;

Var TotalWindows,
    i : Integer;
    Delayer : LongInt;

Begin
  Val(MyDialogData.InputLine,TotalWindows,i);
  If i <> 0 Then NewDialog;
  Delayer:=2 - MyDialogData.CheckBox2;
  If MyDialogData.CheckBox1 = 0
  Then
   Begin
     W:=WLarge;
     H:=HLarge;
   End
  Else
   Begin
     W:=WSmall;
     H:=HSmall;
   End;
  For i:=1 To TotalWindows Do
    Begin
      NewWindow;
      Delay(Delayer*300);
    End;
End;

Procedure TMyApplication.About;

Var
  Dialog: PDialog;
  Control: PView;
  R: TRect;
  C: Word;

Begin
  R.Assign(0, 0, 38, 13);
  Dialog := New(PDialog, Init(R, 'About'));
  With Dialog^ Do
  Begin
    Options := Options or ofCentered;
    R.Grow(-1, -1);
    Dec(R.B.Y, 3);
    Insert(New(PStaticText, Init(R,
      #13 +
      ^C'  Demoprogram TurboVision'#13 +
      #13 +
      ^C' (C) DOS-Special, juni 1991 '#13 +
      #13 +
      ^C'    version 0.98 ')));
    R.Assign(5, 8, 33, 12);
    Insert(New(PButton, Init(R, 'O~K', cmOk,
bfDefault)));
  End;
  C:=Desktop^.ExecView(Dialog);
    Dispose(Dialog, Done);
End;

Var
  MyApplication: TMyApplication;

Begin
  MyApplication.Init;
  MyApplication.Run;
  MyApplication.Done;
End.
```

De overstap naar Clipper 5.0

Wie in dBase programmeert, weet waarschijnlijk dat de afgelopen september gelanceerde Clipper 5.0 een zeer krachtige versie van deze taal is. Maar door de vele complexe mogelijkheden van Clipper 5.0 bent u er misschien nog niet toe gekomen de overstap te maken. En inderdaad, om goed gebruik te maken van alle nieuwe mogelijkheden van deze nieuwe compiler, moet er totaal verschillend tegen dBase-programmeren worden aangekeken. Is dat de moeite waard? Ik denk van wel, en ik zal in dit artikel duidelijk maken waarom.

Het belangrijkste argument is dat Clipper 5.0 programma's sneller, flexibeler en meer modulair maakt. Laten we beginnen met modulariteit. Ter illustratie het volgende fragment:

```
FOR i = 1 TO 100
  DO MyProc
NEXT
```

Hoevaak wordt MyProc aangeroepen? Nee, geen 100 keer. Het antwoord is dat het niet te bepalen is. In standaard dBase (en dus ook Clipper Summer '87) is de variabele i zichtbaar binnen alle procedures. Indien MyProc de waarde van i verandert zou de For/Next-loop volledig worden ontregeld.

Dit wordt veroorzaakt doordat standaard dBase variabelen dynamisch creëert. De levensduur van variabelen duurt totdat ze expliciet vrijgegeven worden of totdat de module waarin de variabele werd aangemaakt, beëindigd wordt. Weliswaar kan een module het PRIVATE-commando gebruiken om zijn variabelen te verbergen voor de aanroepende routine, maar deze routine kan niet controleren of de module dat inderdaad heeft gedaan.

Hieruit volgt dat in dBase een van de hoofdprincipes van modulair programmeren geweld wordt aangedaan: een module mag geen onverwachte neveneffecten hebben in andere delen van het programma. Of, anders gesteld, elke willekeurige procedure of UDF (user-defined function) kan interfereren met andere procedures of functies.

Lexicale scoping

Clipper 5.0 verbetert dit door gebruik te maken van lexicale scoping. Het concept hiervan zal bekend zijn aan C- en Pascal-programmeurs. Variabelen die lexicaal 'gescoped' zijn worden aan de compiler bekend gemaakt en hebben een vastomlijnde scope (zichtbaarheidsgebied) en levensduur. In 5.0 wordt dit bereikt met twee nieuwe storage classes: local en static.

Een lokale variabele is alleen zichtbaar binnen de functie die de variabele declareert. Dus indien we bovenstaand fragment als volgt herschrijven:

```
LOCAL i
...
FOR i = 1 TO 100
  DO MyProc
NEXT
```

dan kan MyProc niet aan de variabele i komen, ook al wordt binnen MyProc ook een variabele i gebruikt.

Het LOCAL statement reserveert geen ruimte voor de variabele i, maar geeft een speciale betekenis aan de naam van de variabele. De variabele wordt gecreëerd zodra de functie waarin hij wordt gedeclareerd begint, en houdt op met bestaan zodra die functie dat doet.

Een STATIC variabele heeft dezelfde scope als een LOCAL, maar blijft tussen de functieaanroepen door wel bestaan, en behoudt dus

Niet alle gelijken zijn gelijk

Clipper ondersteunt nu drie verschillende gelijkheidsoperatoren: de enkele operator (=), te gebruiken voor zowel toekenningen en vergelijkingen; de dubbele operator (==), alleen te gebruiken voor vergelijkingen; en de := operator, te gebruiken voor in-line toekenningen.

Een in-line toekenning is verschillend van een reguliere toekenning omdat een in-line toekenning een resulterende waarde oplevert. Dus a:=b maakt niet alleen a gelijk aan b, maar de gehele uitdrukking krijgt ook nog eens deze waarde. Dit betekent dat een in-line toekenning gebruikt kan worden zoals de volgende voorbeelden illustreren:

```
b:=10
c:=(a:=b) * 5  && a==10, c==50
p:=q:=r:=100   && alledrie 100
? Vandaag:=Date()
```

Nantucket adviseert om in het vervolg := voor alle toekenningen te gebruiken, en == voor alle vergelijkingen. De = operator wordt alleen voor compatibiliteit met bestaande toepassingen ondersteund.

zijn waarde. STATIC variabelen die voor de eerste functie of procedure in een .PRG-bestand worden gedeclareerd, hebben dat bestand als hun scope. (Wees hier voorzichtig mee; de compiler genereert immers automatisch een main procedure bovenin de sourcefile. Om statische variabelen met een scope van het gehele bestand te gebruiken, moet men compileren met de /N-optie.)
Tijdens de declaratie kunnen nu ook waarden aan lokale en statische variabelen worden toegekend:

```
LOCAL VarA := 100, VarB := Cmonth(Date())
STATIC VarC := 10, VarD := "Hallo"
```

(Indien u nog niet bekend bent met de := operator, lees dan het tekstkader 'Niet alle gelijken zijn gelijk'.) Bij lokale variabelen wordt de waarde berekend tijdens het draaien van het programa (at run-time) en hier mag dus elke geldige Clipper-uitdrukking worden gebruikt (dit geldt ook voor public en private variabelen). Statische variabelen worden gecreëerd en dus geïnitialiseerd tijdens compilatie, waaruit volgt dat de waarde moet bestaan uit constanten of eenvoudige uitdrukkingen die de compiler kan evalueren.

Ook kunnen er lokale parameters in één procedure of functie gebruikt worden:

```
FUNCTION MyFunc(x,y,z)
```

of

```
PROCEDURE AnyProc(Param1,Param2)
```

Natuurlijk kan ook de oude manier met het PARAMETER-commando nog gebruikt worden, maar hierbij worden private in plaats van lokale parameters gecreëerd. De twee methoden kunnen niet binnen een procedure of functie tegelijk gebruikt worden.

Lexicale scoping maakt het ook eenvoudiger om functies te schrijven die optionele of default argumenten hebben. Beschouw bijvoorbeeld een functie Mess die een bericht afdrukt, met coördinaten die optioneel zijn.

Bijvoorbeeld:

```
Mess(MyMessage)
```

voor de default coördinaten, en

```
Mess(MyMessage,23,50)
```

voor een bericht met coördinaten 23 en 50.

Om dit te kunnen doen, moeten we de functie Mess als volgt laten beginnen:

```
FUNCTION Mess(message,r,c)
  MessRow := IF(r==NIL,24,r)
  MessCol := IF(c==NIL,0,c)
```

Indien de aanroepende routine een argument weglaat, krijgt de corresponderende parameter de waarde NIL. De aangeroepen routine kan hierop testen zoals te zien is in het voorbeeld. Merk op dat NIL zowel een waarde als een data-type is. Lokale en statische variabelen hebben het type NIL totdat hun een waarde wordt toegekend (waaruit dan

ook het type kan worden afgeleid).

Naast het verbeteren van modulariteit, hebben variabelen met lexicale scoping nog een ander groot voordeel: ze zijn veel efficiënter. Tijdens uitvoering van het programma zijn al hun namen reeds door de compiler geconverteerd naar adressen, dus het run-time besturingsprogramma hoeft niet in tabellen te zoeken naar hun waarden. Dit kan een opmerkelijke tijdwinst opleveren! Een eenvoudige benchmark (listing 1) laat zien hoe het vervangen van PRIVATE door LOCAL variabelen een 37% verbetering in executie-tijd oplevert.

Clipper

Indien u besluit om over te stappen op lokale en statische variabelen, zijn er enkele punten om op te letten. De functie Type() kan niet met dergelijke variabelen gebruikt worden (in plaats hiervan moet ValType() gebruikt worden), en ook ReadVar() kan niet worden gebruikt. Bovendien kunnen ze niet in .MEM-bestanden bewaard worden, kunnen hun namen niet voor macro-substitutie worden toegepast. Als hun namen wel bekend moeten zijn aan de debugger, moet de /B compiler-switch worden gebruikt.

Arrays

Lexicale scoping heeft ook implicaties voor arrays. Een array kan als local of static gedeclareerd worden:

```
LOCAL Arr1[100], Arr2[10,2]
STATIC Arr3[5]
```

Natuurlijk blijven public en private arrays ondersteund. Het DECLARE-commando kan nog steeds gebruikt worden om een private array te definiëren, maar Nantucket raadt dit af.

Op het eerste gezicht lijken arrays in 5.0 wat verwarrend, met name omdat ze afwijken van de conventies die in andere talen gebruikt worden. Het helpt wellicht om drie punten te onthouden. Ten eerste is een array een data-type op zichzelf. De functie ValType(Arr1) keert dan ook "A" (voor Array) terug, ongeacht de typen van de individuele elementen.

Zoals met de meeste andere data-typen kan de waarde van een array eveneens als een constante worden weergegeven:

```
LOCAL WorkDays[5] :=
  { "Ma", "Di", "Wo", "Do", "Vr" }

MyArray := { 5, 10, 15, 20 }

Result := MyArray[2] *
  { 100, 150, 250 }[3]    && geeft 2500
```

Accolades worden gebruikt om de waarden mee af te bakenen, en rechte haakjes voor indexeringen. Zoals het laatste voorbeeld laat zien is het geen probleem een array 'on the fly' aan te maken en te indexeren.

Arrays kunnen in Clipper 5.0 multi-dimensionaal zijn:

```
LOCAL MultArray[4,3]
:= { { "Winter", "Lente", "Zomer",
       "Herfst" },
     { 5, 10, 15, 20 },
     { .F., .T., .F., .T. }
   }
? MultArray[3,2]        && drukt 15 af
```

Eigenlijk is het enige verschil met andere talen het feit dat arrays elementen van verschillende datatypen kunnen bevatten. Maar in Clipper 5.0 zijn multi-dimensionale arrays niet beperkt tot rechthoekige tabel-achtige structuren. Ze kunnen aanzienlijk complexer zijn.

Om dit te begrijpen, moet het tweede punt in gedachten worden gehouden: arrays worden altijd 'by reference' behandeld. Een array-referentie lijkt wel een beetje op een pointer. Hij kan in een variabele opgeslagen worden, als parameter worden meegegeven aan een functie en teruggegeven worden door een functie. Dus in het volgende statement:

```
NewArray := OldArray
```

zal NewArray niet een kopie of kloon van OldArray bevatten, maar zullen beide variabelen referenties bevatten naar dezelfde data.

Omdat referenties in een variabele kunnen worden opgeslagen, kunnen ze natuurlijk ook in een array worden opgeslagen:

```
LOCAL Arr1[4], Arr2[2]
Arr1[1] := "A"
Arr1[2] := "B"
Arr1[3] := "C"
Arr2 := { "D1", "D2" }
Arr1[4] := Arr2
```

Dit is precies hetzelfde als:

```
Arr1 := { "A", "B", "C", { "D1", "D2" } }
```

Hier is een array niet langer een simpele tabel. Drie van de vier elementen bevatten strings, maar het vierde element een andere array. Op deze manier kunnen geneste structuren van willekeurige complexiteit opgebouwd worden.
Clippers array-referenties verschillen iets van pointers in C. In Clipper bestaat de array zo lang als er referenties bestaan. Zodra de referenties ophouden te bestaan, zal de 'garbage collector' van Clipper de ruimte van het array wederom vrijgeven. Ook zijn array-referenties onafhankelijk van het type van het array. Inderdaad, array-types zijn vrij zinloos in Clipper. Je kunt niet echt spreken van arrays van strings of arrays van getallen. Arrays zijn van het type array.

Clippers geneste arrays zijn duidelijk zeer krachtig, maar moeten met zorg worden toegepast. Om dit te illustreren: het is volledig legaal om Arr1[4,1] aan te spreken (hetgeen "D1" oplevert), hoewel het array één-dimensionaal was. Maar bijvoorbeeld Arr1[1,1] levert een run-time foutmelding op.
Met de dubbele gelijktekens kunnen twee arrays vergeleken worden:

```
? NewArray == OldArray
```

Dit levert alleen .T. op indien beide variabelen (referenties) naar hetzelfde array in het geheugen refereerden. Er worden dus geen twee losstaande arrays met elkaar vergeleken. Een ander subtiliteit: de ingebouwde ACopy() en AClone() functies kopiëren beide elementen van de ene array naar de andere. Als de source array sub-arrays bevat zal ACopy() enkel referenties naar die sub-arrays kopiëren, terwijl AClone() de sub-arrays zelf zal kopiëren.

Het derde belangrijke punt over arrays is dat ze helemaal dynamisch zijn, dat wil zeggen dat ze onder controle van het programma kunnen groeien en krimpen. De AAdd() functie voegt een enkel element toe aan het einde van een array, terwijl ASize() de lengte van een array verandert in een bepaald aantal elementen.

Listing 2 laat een user-defined function zien die dynamisch een array creëert. De functie is een eenvoudige generieke database-zoekfunctie, en geeft een lijst van record-nummers terug van de records in de huidige database waarvan een bepaald veld een bepaalde waarde heeft. Het feit dat de teruggegeven waarde een array is geeft geen problemen - het is in feite een referentie. Om dezelfde reden geeft het gebruik van arrays als parameters van UDF's geen problemen.

Clipper 5.0 bevat verschillende ingebouwde functies die op deze wijze betrekking hebben op arrays. DBCreate() creëert een database-structuur gebaseerd op een twee-dimensionale array met de namen, typen, lengten en aantal decimalen van de velden. DBStruct() doet het omgekeerde: deze functie geeft een array terug met alle veldinformatie. Tenslotte geeft Directory() een array terug met informatie over de bestanden in een directory.

Code-blocks

Een andere krachtige innovatie in 5.0 is het code-block, om user-defined commands mogelijk te maken. Het code-block is zelf echter ook een handig hulpmiddel, waarvan geen equivalent bestaat in de meeste andere programmeertalen.

Een code-block is een klein stukje uitvoerbaar programma-code, dat in een variabele kan worden opgeslagen, meegegeven kan worden als een parameter, teruggekeerd kan worden door een functie en in het algemeen zich kan gedragen als elk ander stuk data. Een code-block is dan ook een data-type, net

als een character-string of een datum. Het heeft eveneens de eigenschappen van een user-defined functie, en heeft ook wel iets weg van een macro, hoewel het geen van beide is.

De formele syntax van een code-block is als volgt:

```
{ |[argument list]| <EIXSO>
  list }
```

Enkele voorbeelden:

```
{ |a,b,c| 2*(a+b)/2 }
{ |ErrNo| "Error number"+Str(ErrNo,3)+;
          "detected" }
{ |p,q,r| Func1(p), Func2(q), Func3(r) }
{ || MainFunc() |
```

Zodra het code-block uitgevoerd wordt, wordt elke uitdrukking rechts van de vertikale lijnen geëvalueerd, gebruik makend van de variabelen in de argumentenlijst. De waarde van de meest rechter expressie wordt dan de waarde van het code-block zelf. Merk op dat de vertikale lijnen nodig zijn, zelfs als er geen argumenten zijn. Dit om een code-block van een array-constante te onderscheiden.

Omdat code-blocks toegekend kunnen worden aan variabelen en meegegeven kunnen worden als parameters aan functies, is het volgende fragment toegestaan:

```
MyBlock := { |a,b| MyFunc(a,b) }
x := MyUDF( {|a,b| MyFunc(a,b) } )
```

In geen van beide statements wordt het code-block uitgevoerd. Dit kan gebeuren met de ingebouwde functie Eval(), die als parameters het code-block heeft, gevolgd door de argumenten:

```
MessBlock := { |d| QOut("Vandaag:";
                +CDOW(d)) }
EVAL(MessBlock,DATE())
```

Dit zal er voor zorgen dat een bericht wordt afgedrukt met de dag van de week (De QOut-functie is vergelijkbaar met het "?" commando.)

Code-blocks zijn verschillend van macro's omdat ze tijdens compilatie worden gecompileerd, waardoor ze veel efficiënter zijn. Dit betekent ook dat ze toegang hebben tot variabelen met lexicale scope. Om exact te zijn is de scope van local en static variabelen niet alleen de functie of procedure die ze definieert, maar ook de code-blocks gedefinieerd binnen die functie of procedure.
Waar kunnen we code-blocks toepassen? Een goed voorbeeld is het generaliseren van een functie of procedure. Normaal gesproken worden aan een UDF data meegegeven waarop vervolgens handelingen worden verricht. Met code-blocks kan van buitenaf de wijze waarop deze handelingen verricht worden, beïnvloed worden.

Als voorbeeld een eenvoudige functie die in een array zoekt naar een opgegeven waarde, die als volgt gebruikt kan worden:

```
IF Scan(Array1,"Lewis")
  ? "Gevonden"
ENDIF
```

De parameters voor Scan() zijn de namen van het array en de gezochte waarde. De functie keert .T. terug indien de waarde gevonden wordt.
Het zou aardig zijn als we deze functie konden generaliseren, zodat in plaats van een exacte overeenkomst, ook gezocht kon worden met bijvoorbeeld een Soundex-overeenkomst, of onafhankelijk van hoofd- en kleine letters. Een manier om dit te bewerkstelligen zou kunnen zijn door een code mee te geven, die de zoek-methode zou aangeven. Maar hierbij zou elke nieuwe zoek-methode leiden tot een wijziging van de Scan-functie.

Een betere oplossing is om de methode in een code-block mee te geven. Die wordt vervolgens aan de UDF meegegeven net als elke andere parameter:

```
Scan(Array1, { |x|
Soundex(x)==Soundex("Lewis") } )
```

of:

```
Scan(Array1, { |x| "Lewis" $ x } )
```

Mike Lewis

```
* LISTING 1

* Kleine benchmark om local en private
* variabelen met elkaar te vergelijken

PRIVATE p1, p2:=1, p3:=1
LOCAL l1, l2:=1, l3:=1
StartP:=Seconds()

FOR p1=1 TO 4000
  p2 += p3
NEXT
StartL:=EndP:=Seconds()

FOR l1=1 to 4000
  l2 += l3
NEXT
EndL:=Seconds()

? "Tijden zijn als volgt:"
? "     Met private variabelen:";
                  +STR(EndP-StartP,5)
? "     Met local   variabelen:";
                  +STR(EndL-StartL,5)
RETURN
```

```
* LISTING 2
FUNCTION GetRecs
PARAMETERS fld, contents
* Algemene database zoek-functie; geeft
* recordnummers terug van elk record dat
* voldoet aan de zoekvoorwaarde

LOCAL recs := {}             && Leeg array
LOCATE FOR &fld. == contents
DO WHILE found()
  AAdd(recs,Recno())
  CONTINUE
ENDDO
RETURN recs
```

```
* LISTING 3
FUNCTION Scan(Ar,Blk)
* Functie om in een array te zoeken naar
* een bepaalde voorwaarde.
LOCAL i
FOR i=1 TO Len(Ar)
  IF Eval(Blk,Ar[i])
    RETURN .T.
   ENDIF
 NEXT
 RETURN .F.
```

```
* LISTING 4
FUNCTION GetRecs1(fld, ConditBlk)
* Gewijzigde generieke database zoek-
* functie
LOCAL recs := {}
GOTO TOP
DO WHILE .not. eof()
   IF Eval(ConditBlk,fld)
    AAdd(recs,Recno())
  ENDIF
  SKIP
ENDDO
RETURN recs
```

```
* LISTING 5
FUNCTION DataScan(Operation,
                  ForBlk,WhileBlk)
* Voert een aktie uit op elk record dat
* geselecteerd is met For en While
DO WHILE Eval(WhileBlk) .and. .not eof()
  IF Eval(ForBlk)
    Eval(Operation)
  ENDIF
  SKIP
ENDDO
RETURN Nil
```

Beveiliging van de harde schijf

Regelmatig lezen we in de krant over krakers, die gegevens stelen van andere computergebruikers. Nu zullen uw gegevens niet worden gestolen door krakers, maar ze kunnen wel gelezen worden door andere mensen. Als u op het werk met een computer werkt, zult u het niet prettig vinden als een collega de gegevens leest. Het kan ook, dat de kinderen een spelletje willen spelen en per ongeluk gegevens wissen. Het is dus van belang dat de harde schijf wordt beveiligd.

De meest gebruikte en waarschijnlijk ook de meest doeltreffende beveiliging is een wachtwoord. Het is erg belangrijk op welk punt de computer om een wachtwoord vraagt. De eerste keer dat de PC van de harde schijf leest is om de bootsector te laden. Dus wat is er beter dan op dit punt om het wachtwoord te vragen?

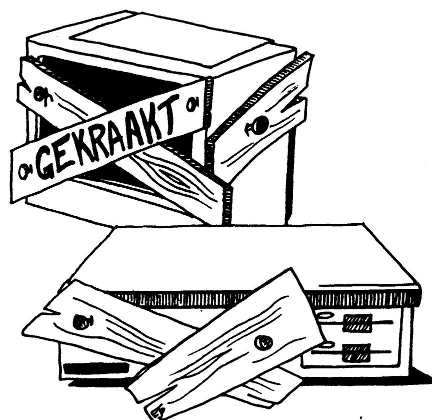
De werking

Normaal gesproken probeert de PC eerst de bootsector van drive A te lezen. Lukt dit niet, dan probeert hij drive C. Heeft hij dit geladen, dan geeft hij de controle over aan het programma dat in deze bootsector staat. In een orginele bootsector van de harddisk staat de Partition Table, de verdeling van de harde schijf.

Bij de PC begint de eerste partitie op kant 1, cilinder 0, sector 1, terwijl de Partition Table op kant 0, cilinder 0, sector 1 staat. Er is dus een aantal sectoren vrij (om precies te zijn 16 sectoren). Bij deze beveiliging zetten we het programma van het wachtwoord in sector 1, en de Partition Table in sector 2. De PC start op, laadt sector 1, en vraagt om een wachtwoord. Als dit juist is ingevoerd, zal hij sector 2 laden en verder gaan met het opstarten.

Nu is er nog een probleem: als men van drive A naar drive C wisselt, leest de computer sector 1 om de partitie te bekijken van drive C. In ons geval staat er dus geen partitie, maar een wachtwoordprogramma. De PC moet sector 2 lezen, niet sector 1. Hoe vertellen we hemdat? Het lezen van schijf gaat allemaal via interrupt 13h. Deze interrupt kunnen we omleggen naar een programma in het geheugen. Dit programma controleert of de PC sector 1 wil lezen. Is dit het geval, dan moet hij sector 2 lezen. Ten slotte moet de leesopdracht worden uitgevoerd, via de oude interrupt 13h.

Tot nu toe hebben we het aardig globaal gehouden. Maar er zijn een paar details die nog besproken moeten worden. Ten eerste het omlegprogramma. Dit moet altijd in het geheugen aanwezig zijn. We kunnen de totale geheugenruimte verlagen met een kilobyte, maar dit zou zonde zijn. We hebben slechts 17h bytes nodig. Nu heeft DOS een ruimte van 60 bytes gereserveerd voor gegevensuitwisseling tussen twee programma's (de Intra Applications Communication Area), die nooit wordt gebruikt. Het adres is 0000:04ACh.



Ten tweede het laden van sector 2. Door het terugkeeradres op de stapel te zetten, zorgen we ervoor dat int 13h terugspringt naar 0000:7C00h, waar op dat moment de origi-

nele Partition Table staat.

Het derde punt is het terugspringen van int 13h. Als deze interrupt wordt aangeroepen staat het terugkeeradres op stapel. Door naar de oude int 13h te JuMPen in plaats van te CALLen zorgen we ervoor dat int 13h automatisch terugspringt naar de plaats vanwaar hij werdt aangeroepen.

Als u het wachtwoord heeft geïnstalleerd op uw harde schijf, en opstart met een bootable disk in drive A, zult u zien dat drive C niet toegankelijk is. Dit is een belangrijk pluspunt van dit programma. Men kan alleen drive C benaderen door op te starten van drive C. Op deze manier kan men niet om het wachtwoord heen. Kinderen kunnen zo wel spelletjes spelen die op diskette staan, maar ze kunnen de harde schijf niet benaderen.

Het opstarten

Het gebruik van het wachtwoord is vrij simpel. Men moet het wachtwoord intypen en op return drukken. Op het scherm verschijnen blokken, zodat niemand kan zien wat u intypt.

Bij installatie is het wachtwoord "verander"! Dus de eerste keer dat u de computer opstart, zult u dit moeten intypen. Natuurlijk moet u dit wachtwoord wel veranderen in uw eigen wachtwoord. Als u het wachtwoord wilt wijzigen, moet u het oude wachtwoord intypen, gevolgd door een schuine streep naar rechts, de slash ("/"). Hierna typt u het nieuwe wachtwoord in, gevolgd door return. De computer zal nu om een bevestiging vragen, om type-fouten te voorkomen. Na het nieuwe woord opnieuw te hebben ingetypt, moet u op return drukken. Het wachtwoord is nu gewijzigd. De volgende keer dat u opstart, zal de PC om het nieuwe wachtwoord vragen.

Er zijn meerdere manieren om het wachtwoord te installeren. U moet eerst PASSWORD.ASM compileren met bijvoorbeeld TASM. Daarna moet u het linken, en tenslotte er een .COM-file van maken. Dit kan door:

```
TASM PASSWORD
TLINK PASSWORD /t
```

```
MASM PASSWORD
LINK PASSWORD
EXE2BIN PASSWORD PASSWORD.COM
```

Vervolgens zijn er verschillende mogelijkheden. U kunt de listings overnemen en PASSINST.BAT starten. Maar u kunt ook Norton Utillities gebruiken.

Ik raad u aan om de listings over te nemen, en deze op een opstartbare diskette te zetten, samen met PASSWORD.COM en het DOS programma DEBUG. Zo kunt u de beveiliging gemakkelijk opheffen (d.m.v. PASSREMO.BAT).

Er zijn dus twee programma's voor u van belang, namelijk:

- PASSINST.BAT : installeert het wachtwoord op de harde schijf.
- PASSREMO.BAT : verwijdert het wachtwoord.

Dit programma is een goede beveiliging van uw harde schijf. Het is een beveiliging die geen geheugen-ruimte kost en geen ruimte inneemt op uw harde schijf. Ten slotte is hij moeilijk te kraken, zelfs voor experts. Genoeg redenen om uw harde schijf te voorzien van een wachtwoord.

Barry Lagerweij

;PASSWORD.ASM

```
Code    segment public
        assume  cs:Code, ds:Code
        org 100h

Vektor = this word -100H+4CH
      ; De oude int 13h vektor
Place   equ 4ach        ; Plaats van nieuwe int13h
DriveC  equ 80h         ; Aanduiding voor drive C
Read1   equ 0201h       ; Lees 1 sector

start: jmp short begin; Ga naar het begin

;
----------------------------------------------
; - Procedure: Prtchar
; - Doel      : Schrijft een string naar het
scherm
; - Input     : si=offset van string
; - Gebruikt : Int 10h functie 0Eh
; -            Reg ax,bx
;
----------------------------------------------
prtchar:
        mov ah,0EH          ; Functie 0Eh, schrijf kar
        mov bx,7            ; Pag.0, attr. 7
loop1:
        lodsb               ; Laad [si] = al, si=si+1
        not al              ; Sleutel!
        or al,al            ; Nul? =einde string
        jz short exit       ; Zo ja, stoppen
```

```
        int 10h          ; Video, schrijf kar. al
        ;   teletype mode
        jmp short loop1 ; Volgende karakter
exit:
        ret              ; Terug

;
------------------------------------------------
; - Procedure: Skip
; - Doel     : Slaat een string over
; - Input    : si=offset string
; - Gebruikt : Niets
; -
;
------------------------------------------------
skip:
        lodsb            ; Laad [si]=al,si=si+1
        not al           ; Sleutel!
        or al,al         ; Is ah reeds nul
        jnz short skip   ; Zo niet, ga verder
        ret              ; Anders terug

;
------------------------------------------------
; - Procedure: Askkey
; - Doel     : Leest een string van toetsenbord
; - Input    : di=offset bestemming string
; - Gebruikt : Int 10h, functie 0Eh, int 16h,fc.
0
; -            Reg ax,cx
;
------------------------------------------------
askkey:
        xor ah,ah        ; ah=functie nul, lees kar
        int 16h          ; Lees ingetypte karakters
        ; karakter in reg al
        cmp al,0Dh       ; Return ingedrukt?
        je short prenter; Ja ?, spring verder
        cmp al,8         ; Backspace ingedrukt?
        jne short writekey
        ; Zo niet,schrijf karakter
        cmp di,si        ; Niet aan begin string?
        je short askkey ; Begin?=volgende toets
        dec di           ; Anders di=di-1
        mov ah,0Eh       ; Schrijf backspace
        int 10h          ; Video, schrijf kar. al
        ;   teletype mode
        mov ax,0A20h     ; Schrijf zonder cursor
        mov cx,1         ; 1 x Spatie
        int 10h          ; Video, schrijf kar. al
        ;   en beweeg cursor
        jmp short askkey; Volgende toets
writekey:
        not al           ; Pas sleutel toe
        stosb            ; Schrijf naar geheugen
        mov ax,0EB0h     ; Schrijf kar B0 ()
        int 10h          ; Video, schrijf kar. al
        ;   teletype mode
        jmp short askkey; Volgende toets
prenter:
        xor al,al        ; Enter gedrukt, schrijf 0
        not al           ; Sleutel
        stosb            ; Schrijf naar geheugen
        ret              ; Ga terug

begin:
        xor ax,ax        ; ax=segment 0
        mov ss,ax        ; stacksegment=0
        mov ds,ax        ; datasegment=0
        mov bx,7C00h     ; terugkeeradres=bx
        mov sp,bx        ; stackpointer op 7C00
        pushf            ; Push flags
        push ax          ; Push terugkeer segment
        push bx          ; Push terug offset(7C00)
        les ax,dword ptr Vektor
        ; Oude int 13h
        mov OldInt,ax    ; Bewaar int. op jumpadres
        mov Oldint+2,es ; offset^ en segment
        mov Vektor,Place; Nieuwe vektor omleggen
        mov Vektor+2,0   ; offset^ en segment
```

```
        push cs          ; es=cs
        pop es
        mov si,offset Mark
        ; maak klaar voor kopieren
        mov cx,OldInt-Mark+4
        ; si=offset nieuwe vektor
        mov di,Place     ; cx=lengte; di=plaats
        cld              ; Kopieer vooruit
        rep movsb        ; Kopieer totdat cx==0

prtpass:
        mov si,offset OldInt+4
        ; si=offset "Password: "
        call prtchar     ; afdrukken op scherm
        call skip        ; sla tekst over ("Check")
        mov dx,si        ; dx=si=offset huidig ww.
        call skip        ; si=offset getypte ww.
        mov di,si        ; di=si=getypte wachtwoord
        call askkey      ; di=einde getypte woord
        mov di,dx        ; di=huidig wachtwoord
        mov cx,si        ; cx=si=ingetoets woord
        sub cx,di        ; cx=si-di=lengte wachtw.
        cld              ; Vooruit
        repe cmpsb       ; Vergelijk tot cx=0 of
        ; tot er een verschil is
        jz short okpass ; Zelfde? ga naar okpass
        dec si           ; si=kar. dat verschilt
        dec di           ; di=di-1
        lodsb            ; Kijk waar verschil zit
        not al           ; Sleutel
        cmp al,2Fh       ; Is het een '/'
        jne short prtpass; Zo niet, begin opnieuw
        mov al,[di]      ; di=einde password
        not al           ; Sleutel
        or al,al         ; Echt hele wachtwoord?
        jne short prtpass; Niet goed? opnieuw
        push si          ; Push veranderde wachtw.
        call skip        ; Ga naar einde
        mov di,si        ; di=begin controle
        push di          ; di=begin controle
        mov si,offset OldInt+4
        ; si=text "Password"
        call skip        ; Sla over
        call prtchar     ; Print "   Check: "
        call askkey      ; Vraag om controle
        mov cx,di        ; cx=di=einde controle
        pop di           ; di=begin controle
        pop si           ; si=begin veranderd woord
        mov bx,di        ; bx=di=begin controle
        sub cx,di        ; cx=lengte controle
        mov ax,cx        ; Tijdelijk lengte bewaren
        cld              ; Richting=vooruit
        repe cmpsb       ; Controleer
        jz short next    ; Ok? ga verder
        jmp prtpass      ; Niet goed? begin opnieuw
next:
        mov di,dx        ; di=dx=begin oude wachtw
        mov si,bx        ; si=bx=begin controle
        mov cx,ax        ; cx=ax=lengte controle
        cld              ; Vooruit
        rep movsb        ; Kopieren
        xor al,al        ; al=0, rest wegvegen
delold:
        stosb            ; Schrijf nul
        mov ah,[di]      ; Reeds einde?
        or ah,ah         ; Is nul?
        jnz short delold; Zo niet, ga door
        mov ax,301h      ; Schrijf sector
        mov cx,1         ; Een sector, track 0
        mov dx,DriveC    ; Schijf C
        mov bx,7C00H     ; Offset van dit programma
        int 13h          ; Disk C, ah=func 03h
        ;   schrijf sectors.
okpass:
        mov ax,0E0Dh     ; Schrijf een Return
        mov bx,7         ; Pag 0, kleur 7
        int 10h          ; Video, schrijf kar. al
        ;   teletype mode
        mov al,0Ah       ; Schrijf Line Feed
        int 10h          ; Druk af op scherm
```

```
        mov ax,Read1     ; Lees een sector
        mov cx,2         ; = Orginele bootsector
        mov dx,DriveC    ; Drive C, Track 0
        mov bx,7C00h     ; Laden over dit programma
        push es          ; Alle vlaggen wissen
        popf             ; Pop flags
        jmp short JmpOld; Ga naar oude int 13h

Mark        = this byte+7b00h; Merkteken
blokkopie

        cmp ax,Read1     ; Een sector lezen?
        jnz JmpOld       ; Zo niet, exit
        cmp cx,1         ; Lees track 0, sector 1?
        jnz JmpOld       ; Zo niet, exit
        cmp dx,DriveC    ; Lees van drive C?
        jnz JmpOld       ; Zo niet, exit
        inc cl           ; Lees sector 2 (org.boot)

JmpOld:
OldInt = this word +7b01h
        db 0EAh,0,0,0,0  ; Jump Far naar oude int.

        db 0F2H,0F5H,0AFH,9EH    ; "Password: "
        db 8CH,8CH,88H,90H,8DH   ;   in code
        db 9BH,0C5H,0DFH
        db 0FFH                  ; einde string

        db 0F2H,0F5H,0DFH,0DFH   ; "Check: "(code)
        db 0DFH,0BCH,97H,9AH,9CH
        db 94H,0C5H,0DFH
        db 0FFH                  ; einde string

        db 89H,9AH,8DH,9EH,91H   ; wachtwoord
        db 9Bh,9AH,8DH           ; = "verander"
        db 0FFH                  ; einde string

        org 2feh                 ; Einde sector
        db 55h, 0AAh             ; Bootsector-merk

Code    ends
        end        start

;EOF PASSWORD.ASM
```

PASSINST.BAT

```
REM PASSINST.BAT
echo [1mPassword 1.01[0m
echo Made by L.A.Software in 1991
if not exist oldboot.com debug getboot.deb nul
debug  inst.deb  nul
echo Installed.   Password = "verander"

REM EOF PASSINST.BAT
```

PASSREMO.BAT

```
REM PASSREMO.BAT
echo [1mPassword 1.01[0m
echo Made by L.A.Software in 1991
if not exist oldboot.com debug getboot.deb nul
debug  remove.deb  nul
echo Removed.

REM EOF PASSREMO.BAT
```

NOLDBOOT.COM

```
l
a300
int 13

rax
301
rbx
100
rcx
1
rdx
80
rip
300
g302
q
```

NOLDBOOT.COM

```
l
a300
int 13

rax
301
rbx
100
rcx
2
rdx
80
rip
300
g302
npassword.com
l
a300
int 13

rax
301
rbx
100
rcx
1
rdx
80
rip
300
g302
q
```

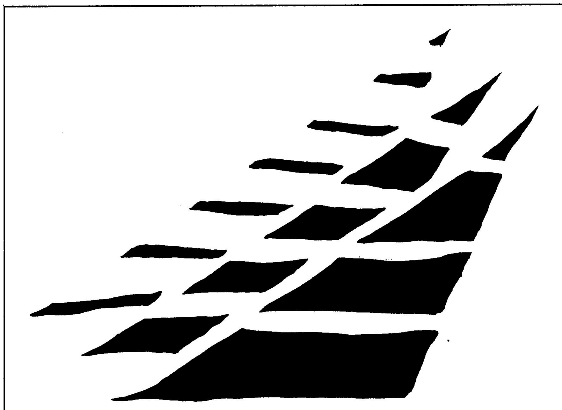
Getboot.deb

```
a300
int 13

rax
201
rbx
100
rcx
1
rdx
80
rip
300
g302
noldboot.com
rcx
200
w
q
```

De DESQview API

Het programma DESQview is vooral bekend als multitasking-programma voor DOS-applicaties. Minder bekend is dat er ook specifieke DESQview-toepassingen kunnen worden geschreven met behulp van een speciale bibliotheek van routines, de DESQview API Libraries geheten. Zulke libraries zijn beschikbaar voor C, Pascal, dBase, Basic en Assembly. In dit artikel zullen we de C-versie bespreken, en enkele korte voorbeeldprogramma's ontwikkelen.



QuarterDeck logo

De DESQview-omgeving

Voor diegenen die het DESQview-pakket niet kennen, eerst een korte omschrijving van de omgeving. QuarterDeck, de producent van DESQview, heeft dit pakket al enige jaren geleden uitgebracht en het is dan ook een van de meest populaire DOS-programma's voor multitasking. Het voordeel van DESQview ten opzichte van MS/Windows is dat ook op computers zonder een 80286- of 80386-processor programma's tegelijkertijd (concurrent) kunnen worden gedraaid. Wordt er geen gebruik gemaakt van multitasking, dan biedt DESQview nog altijd het voordeel van task-switching, hetgeen de vaak tijdrovende 'load-quit-load-quit'-procedure voorkomt. Zo kan er tegelijkertijd in WordPerfect en bijvoorbeeld dBase worden gewerkt, terwijl een communicatie-programma in de achtergrond contact onderhoudt met een mainframe. Het is bekend dat de eisen die Windows aan de configuratie stelt, vrij hoog zijn: een snelle computer (80286 of hoger) met minstens 2 MB intern geheugen en een VGA-scherm is toch wel nodig om efficiënt te kunnen werken. Omdat DESQview teken-georiënteerd is, kan het pakket op een monochroom scherm werken; bovendien is het gebruik van DESQview in de praktijk niet gelimiteerd tot AT- of snellere computers. Een absoluut voordeel van Windows is dat het er veel beter uitziet en dat er gemakkelijker mee te werken is door de vele grafische voorzieningen.

DESQview-specifieke programma's

Programma's die gebruik maken van de routines in de API library zijn meestal DESQview-speficiek en kunnen niet zonder meer onder DOS worden gedraaid. Het voordeel van zulke programma's is dat ze perfect in het DESQview-systeem passen en van alle mogelijkheden gebruik kunnen maken. Zo kunnen DESQview-specifieke programma's met elkaar communiceren door middel van berichten en mail-boxen. Tevens kunnen dergelijke programma's van het DESQview menu-systeem gebruik maken en met het DESQview-moederprogramma communiceren. Het verschil tussen een DOS-applicatie en een DESQview-specifieke applicatie is niet zo groot als bij DOS- en MS/Windows-toepassingen, en de eerlijkheid gebiedt ons te zeggen dat het DESQview-systeem veel minder 'fancy' is dan Windows. Er zijn veel minder commerciële DESQview-toepassingen dan Windows-toepassingen, en de API wordt dan ook het meeste gebruikt om 'custom' applicaties te schrijven. Maar ook simulaties van concurrente processen kunnen goed met behulp van DESQview worden geschreven.

De DESQview API

De DESQview API is een verzameling routines die toegesneden is op het DESQview multitasking-systeem. Zo zijn er routines

voor windows, multitasking, communicatie tussen processen, geheugenbeheer en randapparaten zoals de muis. Hoewel de library in C is geschreven, is de opzet toch duidelijk object-georiënteerd. Er zijn zes verschillende typen objecten: vensters, mailboxen, het toetsenbord, timers, de muis en panels. Naar zulke objecten kunnen berichten worden gestuurd die tussen de verschillende klassen aardig wat overeenkomsten vertonen. Zo zal het bericht 'WRITE' naar een venster-object gestuurd worden om iets in een venster af te drukken; hetzelfde bericht verzenden naar een mailbox betekent dat een bericht wordt gepost. De object-georiënteerde aanpak van de API heeft de aardige bijwerking dat een 'echt' object-georiënteerde schil in C++ relatief eenvoudig voor de API-functies kan worden geschreven.

Een bericht dat naar een object van een bepaalde klasse kan worden gestuurd, wordt in de naam van de functie verwerkt. Zo wordt de functie win_new() gebruikt om het bericht 'NEW' naar een window-object te sturen, mal_new() om dit bericht naar een mailbox-object te sturen, etcetera.

Een eenvoudig programma

Als eenvoudig voorbeeldprogramma zullen we nu een "hello, world"-applicatie ontwikkelen voor DESQview. Allereerst wordt de desbetreffende API-header opgenomen in de #include-lijst:

```
#include "dvapi.h"
```

Vervolgens wordt een functie Init() geschreven, die controleert of het programma vanuit DESQview werd geladen, en of het de juiste versie van DESQview betreft:

```
  /* Heeft minimaal versie 2.0 nodig */
#define VersionRequired  0x200

void Init()
{
     /* Initialiseer DV API */
  int Version = api_init();

  if ( Version  VersionRequired )
  {
    printf("Dit programma vereist "
    "mimimaal DESQview %d.%02d.\n",
    VersionRequired > 8,
    VersionRequired & 0xFF );
    exit( 1 );
  }

    /* Stel extensies in op de */
    /* vereiste versie */
  api_level( VersionRequired );
}
```

Als Init() succesvol wordt beëindigd, is de API klaar voor gebruik. Vervolgens kan dan het hoofdprogramma worden aangeroepen dat in de functie ProgramBody() staat:

```
void ProgramBody()
{
  win_printf( win_me(),
    "Hello, world!" );
}
```

We zien hier dat er niets anders wordt gedaan dan het sturen van het bericht 'PRINTF' naar een window-object. Het eerste argument van een API-functie geeft meestal het object aan waar het bericht naar wordt verzonden. In dit geval is dat echter het venster dat bij elke applicatie automatisch door DESQview wordt geopend. De functie win_me() geeft het huidige window-object terug, in dit geval dus het applicatievenster. Er zou echter even gemakkelijk een nieuw window-object kunnen worden aangemaakt waarnaar het bericht dan zou worden verstuurd.

Uit het bovenstaande voorbeeldprogramma blijkt dat het schrijven van een eenvoudige DESQview-applicatie minder voeten in de aarde heeft dan een vergelijkbare Windows-applicatie. De DESQview API is minder uitgebreid dan die van Windows, maar daarom niet minder krachtig, vooral op het gebied van het beheersen van taken en processen waar we nu verder op in zullen gaan.

Taken en processen

Het multitasking-aspect van DESQview is zonder twijfel het interessantste. Niet alleen kunnen losstaande programma's tegelijkertijd uitgevoerd worden, ook kan een DESQview-specifiek programma taken ofwel 'threads' afsplitsen. Zo kan er bijvoorbeeld een aparte print-routine worden geïnstalleerd, zodat het programma niet hoeft te wachten als er iets wordt afgedrukt. Normaal gesproken moet de 'caller' bij het aanroepen van een functie wachten totdat deze functie eindigt. Met multitasking zoals geboden door de DESQview API kan de caller na het aanroepen meteen doorgaan, terwijl de

aangeroepen functie tegelijkertijd ook wordt uitgevoerd.
Met de introductie van multitasking komt een aantal problemen aan de orde, waaronder het uitvoeren van atomaire operaties. In een multitasking-systeem kan een routine immers op elk moment worden onderbroken door een andere, tegelijkertijd draaiende routine. Sommige operaties moeten echter achter elkaar worden uitgevoerd, zonder dat een onderbreking mogelijk is. Een bekend voorbeeld is het bijwerken van een bestand met gegevens over het saldo van een cliënt van een bank. Als twee routines tegelijkertijd proberen het saldo te wijzigen, kunnen er problemen ontstaan bij de bepaling van het eindsaldo. Stel dat er twee processen A en B zijn die de volgende operaties uitvoeren:

Proces A

1. Haal saldo op
2. Verhoog saldo met 500
3. Schrijf saldo terug

Proces B

1. Haal saldo op
2. Verlaag saldo met 200
3. Schrijf saldo terug

Indien nu het saldo 400 bedraagt, kan in een multitasking-systeem het volgende plaatsvinden:

A.1. Haal saldo op -> 400
A.2. Verhoog saldo met 500
B.1. Haal saldo op -> 400
A.3. Schrijf saldo terug -> 900
B.2. Verlaag saldo met 200
B.3. Schrijf saldo terug -> 200

Het eindsaldo zal in dit geval niet 700 bedragen, maar 200 omdat de operatie B.3 het laatste woord heeft. Het moge duidelijk zijn dat van elk proces de acties achter elkaar, zonder onderbreking moeten worden uitgevoerd. Dus ofwel eerst A en dan B, ofwel andersom. Dit probleem kan in DESQview worden voorkomen door het gebruik van zogenaamde 'critical regions'. Door de acties 1, 2 en 3 van de processen als critical region te beschouwen, kunnen ze niet worden onderbroken en worden ze dus gezien als één atomaire (ondeelbare) operatie. De DESQview-functies api_beginc() en api_endc() geven het begin respectievelijk einde van een critical region aan.

De tsk_class is aanwezig om task-objecten te definiëren. Met tsk_new() wordt een nieuwe task opgestart in een apart venster. Met de berichten STOP en START (respectievelijk de tsk_stop()- en tsk_start()-functies) kunnen taken tijdelijk gestaakt en weer opnieuw opgestart worden.

Applicaties verschillen van taken in het feit dat ze door DESQview gezien worden als aparte toepassingen, waartussen de gebruiker ook kan wisselen. De API bevat de app_ class voor het definiëren van applicaties. Een belangrijke eigenschap van zowel tasks als applicaties is dat ze 'tot hetzelfde programma behoren', ofwel dat ze code- en data-segmenten delen.

Processen hebben in tegenstelling tot taken en applicaties verschillende code- en data-segmenten. Dit betekent dat processen onafhankelijk in het geheugen bestaan, terwijl ze wel alle voordelen zoals interprocescommunicatie hebben. In de DESQview-implementatie zijn processen verschillende programma's, die ook in het DESQview-menu kunnen staan. Dus terwijl een taak in de meeste gevallen een C-functie zal zijn, is een proces een los gecompileerd programma.

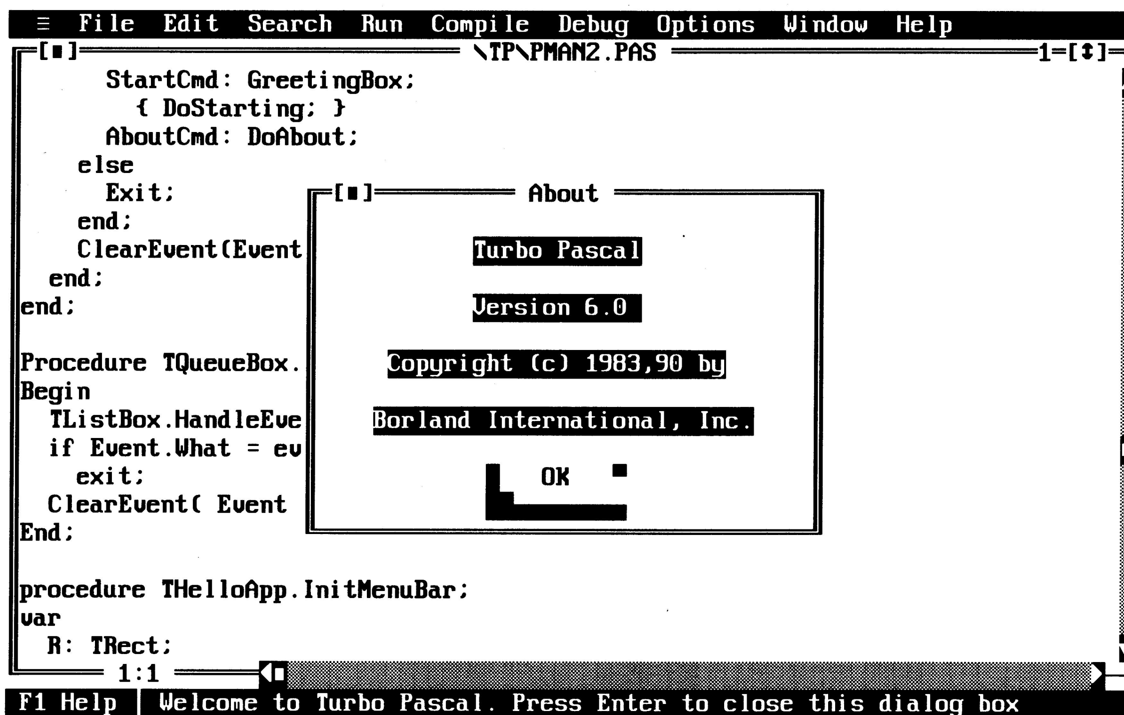
Overige routines

Naast de genoemde klassen bevat DESQview nog ondersteuning voor communicatie tussen processen en taken, het verwerken van berichten veroorzaakt door events, het gebruik van de muis, timers, het toetsenbord en geheugenbeheer inclusief expanded memory-ondersteuning. Het zou echter te ver voeren om al deze mogelijkheden hier te behandelen. Wat is nu het nut van DESQview-specifieke programma's? Het is in ieder geval niet te verwachten dat er zoveel specifieke programma's zullen worden uitgebracht als er 'for Windows'-toepassingen zijn. Het schrijven van DESQview-programma's is eerder beperkt tot educatieve toepassingen (het leren programmeren in een multitasking-omgeving), simulaties en custom-applicaties.

QuickPascal 1.0 versus Turbo Pascal 6.0

Een oneerlijke vergelijking ?

De twee populairste Pascal compilers voor de PC zijn Microsoft's QuickPascal en Borland's Turbo Pascal. De laatste is van deze twee veruit de meestgebruikte en wordt zowel in de amateur- als de professionele hoek aangetroffen. QuickPascal gebruikt men niet zo vaak wanneer het gaat om het schrijven van professionele applicaties. Of dat terecht is leest u in deze dubbeltest van de DOS-Special.

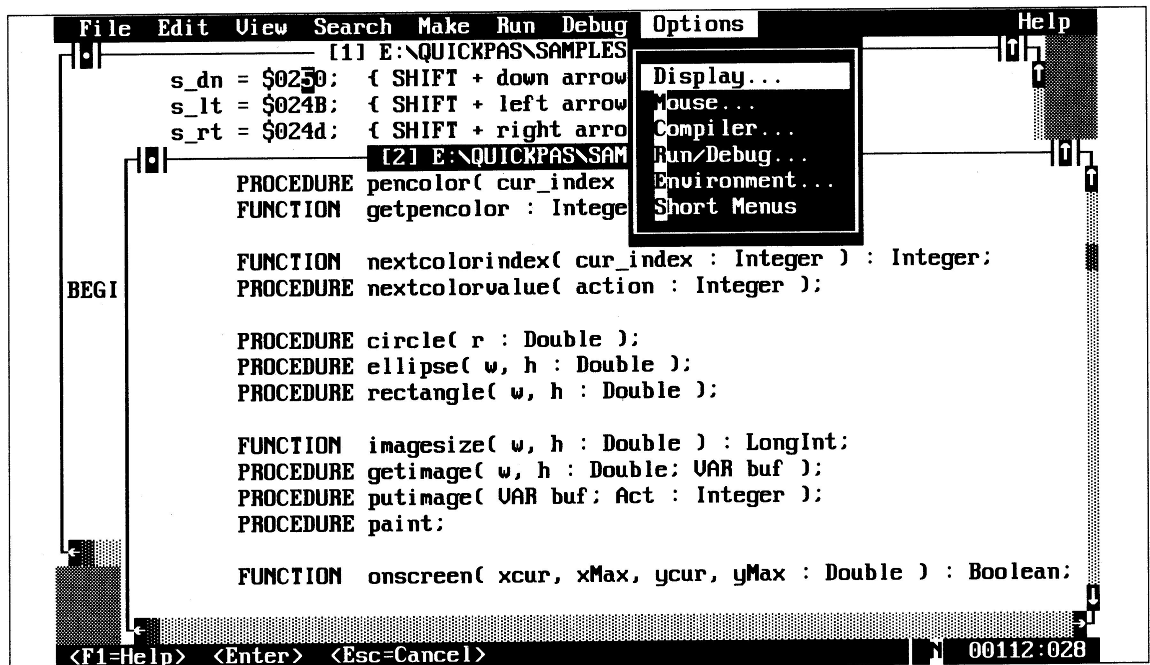


De nieuwe IDE van Turbo Pascal 6.0

Borland levert twee versies van haar Pascal compiler voor DOS (er is sinds kort ook een Windows-hosted compiler, meer hierover elders in dit nummer), de standaard en de professionele editie. Voor deze test gingen we uit van de standaard editie van de laatste versie, 6.0. De professionele editie voegt daar een profiler (besproken in de vorige DOS-Special), een uitgebreidere debugger en de Turbo Assembler aan toe. Microsoft levert van haar QuickPascal geen professionele editie, er is slechts één versie beschikbaar, welke net als Turbo Pascal 6.0 alleen DOS-code (dus geen Windows-code) kan genereren.

Testpunten

Om een vergelijking te maken moesten we de beide compilers op een aantal punten testen. Uiteraard zijn dit zaken als de gebruikersvriendelijkheid van de IDE, de kwaliteit van de debugger, meegeleverde code als units, frameworks en voorbeelden, maar ook de kwaliteit van de handleidingen en het online hulpsysteem. Een ander belangijk punt zijn de toekomstmogelijkheden van eenmaal geschreven code. QuickPascal heet compatibel te zijn met Turbo Pascal, maar verschilt op een voor de toekomst zéér belangrijk punt, de implementatie van object-georiën-



Quick Pascal 1.0

teerd programmeren (OOP). QuickPascal heeft dit gedaan naar de manier die de godfather van Pascal, Niklaus Wirth, heeft bedacht. Borland heeft haar eigen guru, Anders Heijlsberg, op het probleem losgelaten, die met een andere implementatie kwam. Op dit punt is de code dus niet meer uitwisselbaar, zodat u zich twee keer moet bedenken voor u voor een compiler kiest. In de C++ wereld zal zo'n probleem niet snel voorkomen, aangezien AT&T daar de standaard voor maakt waar weinig compiler-fabrikanten van zullen afwijken. Dit garandeert een compatibiliteit die, zeker nu het fenomeen object-georiënteerd zo opduikt, een duidelijke verheldering in de software-wereld betekent. Borland is de ongekroonde koning van Pascal voor de PC, het bedrijf is er groot door geworden en zet zodoende de trend. Wanneer een andere fabrikant een Pascal compiler uitbrengt is het voor de consument het handigst wanneer de code portable is ten opzichte van Borland's 'standaard'.

Maar terug naar waar het allemaal om gaat, TP en QuickPascal. De behuizing van de pakketten gaf al een aardig verschil te zien. QuickPascal kwam in een doos met daarin slechts één handleiding 'Pascal by Example', terwijl TP met maar liefst vier boeken uit de doos kwam. Specifiek voor de compiler waren de 'User's Guide', de 'Programmer's Guide' en de 'Library Reference'. Daarbij kwam nog de 'Turbo Vision Guide', die u het nog te bespreken framework TurboVision moet gaan uitleggen. Als we deze laatste niet in de telling betrekken geeft Borland zo'n achthonderd pagina's aan documentatie mee, terwijl Microsoft het bij één boek van driehonderd bladzijden houdt!

De installatie-procedure was bij beide pakketten simpel, al kreeg ik bij TP op een gegeven moment een foutmelding dat PKUNZIP niet gevonden was. Vreemd, maar uiteindelijk werkte het allemaal wel. De procedure bij QP was wat rommelig; omdat ik niet op de defaultdrive C: wilde installeren werd mij drie keer gevraagd waar wel. Het verschil aan in beslag genomen diskruimte is aanzienlijk. QuickPascal neemt genoegen met 1.8 MB, terwijl TP zo'n 4 MB aan harddiskruimte opeist.

QuickPascal

De IDE van QP valt niet tegen. Microsoft was eerder dan Borland met het toevoegen van een muis en resizable windows aan de ontwikkelomgeving. QP bezit een goede editor, die Pascal keywords in een andere kleur weergeeft. Dit zou de overzichtelijkheid kun-

nen verhogen, ik zelf vond het vrij vervelend, maar misschien moet je eraan gewend raken.

Debugger

De IDE bevat verder een geïntegreerde debugger. De kwaliteit hiervan is zoals men zou verwachten: niet bijzonder spectaculair, maar er valt mee te werken. Zo is backtracing (het 'teruggaan' in het programma) niet mogelijk en laat het debuggen van objectgeoriënteerde code ook te wensen over. De output van de debugger, zoals watch-variabelen en breakpoints, komen in een apart window op het scherm. U kunt de informatie op een voor u handige plaats en grootte op het beeldscherm krijgen. De debugger laat een record-variabele als watch-variabele toe. De datavelden worden door middel van komma's van elkaar gescheiden. Indien u de optie 'R' meegeeft worden ook de namen van de velden getoond. Dit maakt het bekijken van een record-variabele een stuk overzichtelijker. Met de optie '{$D+}' wordt in de sourcecode debug-informatie meegenomen, welke tevens de defaultwaarde is. Aangezien QP conditionele compilerdirectives ondersteund is het mogelijk voorwaardelijke debugstatements toe te voegen.

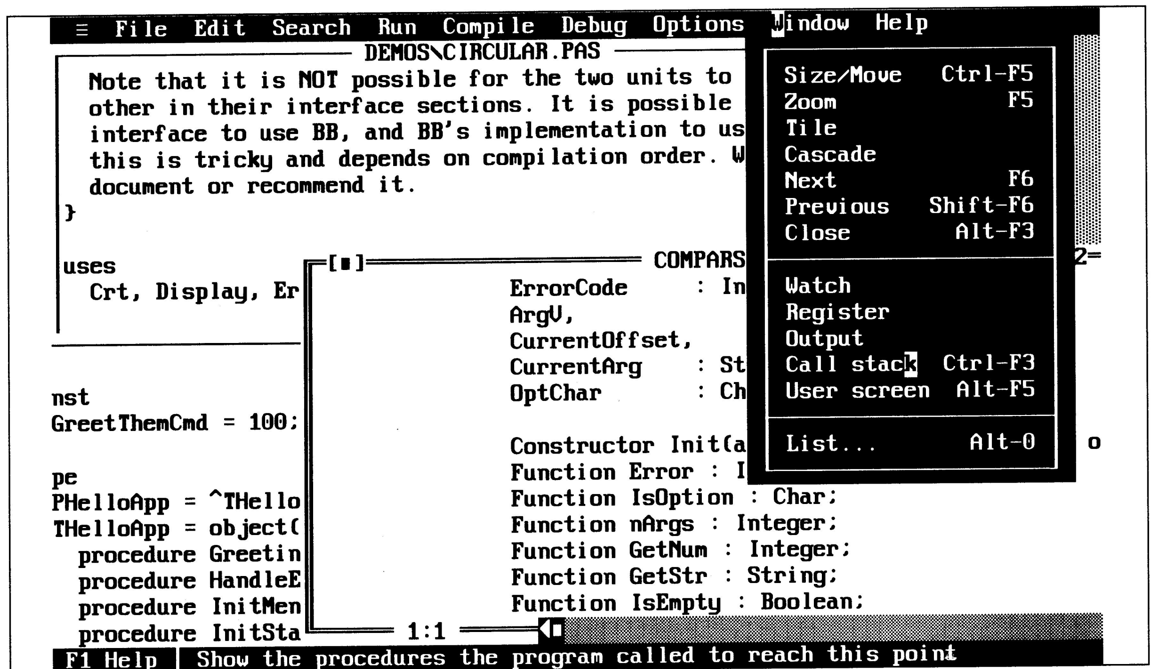
Taal

Microsoft heeft zich bij het ontwikkelen van QP bijna volledig op Borland gericht. Er is geprobeerd om de taal en de units compatibel te houden. Alleen op het gebied van object-georiënteerde code zijn de wegen dus gescheiden geraakt. De compilerdirectives zijn hetzelfde, al zijn het er niet even veel. Gewone code is dus niet voor honderd procent portable, omdat deze compilerdirectives in de source kunnen worden aangegeven. Voor de rest is de code die niet op objecten berust portable te noemen. Wat oude niet objectgeoriënteerde codes die door mij met TP waren geschreven werden zonder meer door QP geaccepteerd. Deze code bevatte een aantal pointeroperaties, maar ook variante records, DOS-functies enzovoorts. Wel werd een unit die in het programma was aangegeven (uses unitnaam) telkens opnieuw gecompileerd, ook indien er geen veranderingen in aangebracht waren. Nu is de compilatiesnelheid van Pascal zo groot dat we niet eindeloos hoeven te wachten, maar toch. In QP zijn in tegenstelling tot TP alle objecten dynamisch. U moet ze dus altijd met 'new' aanmaken en met 'dispose' verwijderen. Nu zou dat nog niet zo'n probleem zijn, ware het niet dat de schrijfwijze van een verwijzing naar velden en methodes van een object niet impliceert dat we met een pointer te doen hebben. Heeft het object 'A' als datafield 'b', waarbij A dus een pointer is, dan roepen we het veld 'b' aan door middel van 'A.b' en niet door 'A^.b'. Aangezien TP deze schrijfwijze voor een pointer-object niet accepteert is de code op dit punt al niet meer uitwisselbaar. Erger nog: in de implementatie van QP zijn alle methoden in een object virtueel zijn. Een constructor en destructor bestaan zodoende niet. Om een methode te overschrijven worden we dan met een ernstig probleem geconfronteerd. De heading van een methode moet namelijk dezelfde zijn als die in het ouderobject (zoals dat ook moet met virtuele methoden in TP). Maar wanneer we dat helemaal niet willen moeten we dus een methode met een nieuwe naam toevoegen aan het object. Dit resulteert in meer methoden per object.

Een ander nadeel van de implementering is dat het niet goed mogelijk is om naar de methoden van een ouder te verwijzen. Een object 'omhoog' is mogelijk door het keyword 'Inherited' te gebruiken, maar de eventuele ouder van dat object valt niet meer te bereiken. Bij gebruik van OOP is in QuickPascal het keyword Self verplicht. In de meeste gevallen is dat overbodig omdat we toch wel weten waar de datamember of methode betrekking op heeft.

Voorbeeldprogramma's in Quick-Pascal

Het aantal voorbeeldprogramma's dat Microsoft meelevert valt op de vingers van een hand te tellen. Een aardig voorbeeld vond ik een grafisch demonstratieprogramma van verschillende sorteermethoden, genaamd sort.pas. Het is jammer dat Microsoft niet



Turbo Pascal 6.0

wat meer mogelijkheden laat zien van Quick-Pascal, omdat er natuurlijk best aardige dingen mee te doen zijn.

Helpsysteem

Microsoft levert bij QuickPascal een uitgebreide tutorial die u wegwijs moet maken in het gebruiken van QuickPascal. Het zou handig zijn als dit in meer programma's beschikbaar zou zijn, omdat het een snelle manier is om een applicatie machtig te worden. Het helpsysteem van QP is net als dat van Turbo Pascal vrij uitgebreid, echter, waar TP een voorbeeld geeft bij het gebruik van een bepaalde functie, laat QP dit achterwege.

Turbo Pascal 6.0

Versie 6.0 van Turbo Pascal is de eerste die een IDE heeft die de compiler waardig is. Beschouwde iedereen versie 5.5 als een goede compiler, de ontwikkelomgeving was op een gegeven moment ouderwets. Gebruik van muis en resizable windows werd niet ondersteund. Met Turbo Pascal 6.0 is dit gelukkig wel het geval. Het aardige van deze IDE is dat die geheel met behulp van Turbo Pascal is geschreven. Zo kunt u zien wat er met het meegeleverde applicatie-framework Turbo-Vision, waarvan u elders in dit blad een uitgebreide beschrijving vindt, mogelijk is. De IDE van TP is nu dus op niveau, en persoonlijk vind ik het iets prettiger werken dan de IDE van QP. Turbo Pascal is in vergelijking tot de vorige versie met een aantal interessante zaken uitgebreid. Zo is er nu ook een commandline-compiler die in protected mode werkt. Voor het gebruik hiervan is minstens een 80286-processor vereist. Deze compiler, genaamd tpcx, genereert dan wel realmode programma's, maar is in staat om extended memory te gebruiken en zodoende zeer grote programma's te compileren. De oude commandline-compiler tpc is natuurlijk gewoon blijven bestaan. Turbo Pascal is nu ook in staat om 286-code te genereren. Specifieke 80286-instructies kunnen nu gebruikt worden om een snellere en efficiëntere code te verkrijgen. Het is jammer dat de bekende DOS 640KB-limiet wel blijft bestaan. Ook heeft TP een nieuwe heapmanager. Volgens Borland veroorzaakt deze minder fragmentatie in het geheugen en is daarbij sneller.

Op het object-georiënteerde vlak zijn de uitbreidingen wat minder spectaculair. De enige nieuwe feature die ingebouwd is is 'private', waarmee datamembers en methoden afgeschermd kunnen worden. De scope van deze datamembers en methoden is dan de module waarin ze gedeclareerd zijn, dus als bijvoorbeeld een methode in een unit x

private gedeclareerd is kan alleen in unit x met deze methode gewerkt worden.

Toegankelijkheid

Borland streeft ernaar om de toetreding tot Pascal ook bij OOP zo makkelijk mogelijk te maken. Zaken als multiple inheritance, repeated inheritance, friends, protected en operator en function overloading zullen niet snel in TP verwerkt worden, al zijn die al wel in talen als C++ te vinden. Nu is dit allemaal niet noodzakelijk om een probleem op te lossen, met het nu aangebodene kan even veel gedaan worden, maar het kan de oplossing af en toe wat vereenvoudigen.

Debugger

De debugger die Borland bij de standaard editie van TP meelevert is net als die van QP niet bijzonder spectaculair te noemen. In de professionele editie wordt de Turbo Debugger meegeleverd die tot veel meer in staat is. De geïntegreerde debugger biedt onder meer conditionele breakpoints en record- en object-watch-variabelen. Ook bij deze debugger geldt dat het niet mogelijk is om terug te gaan in het programma. Wanneer u dan ook prijs stelt op een uitgebreidere debugger doet u er verstandig aan om de professionele editie van TP met de Turbo Debugger aan te schaffen.

Taal

Zoals gezegd is de uitbreiding van TP in OOP-gebied beperkt gebleven tot 'private'. Op het niet-OOP-vlak zijn er echter ook wat zaken bijgekomen. Er is een zogenaamde 'extended syntax directive' bijgekomen ($X), dat het mogelijk maakt om functies te beschouwen als procedures. Het functieresultaat kan dan genegeerd worden. Het compilerdirective {$G+} genereert 286-code. Ook is de inline assembler uitgebreid. Code, die u geschreven heeft met eerdere versies van Turbo Pascal, compileert ook nu weer zonder problemen. De unit Graph3 zorgt weer voor compatibiliteit met de grafische functies van de oudere compilers. Echt veel nieuws is er wat dat betreft dus niet onder de zon, maar de taal is wel wat uitgebreider dan die van QP.

Voorbeeldprogramma's

Over de voorbeeldprogramma's die Borland bij Turbo Pascal meelevert kunnen we niet klagen. De kwaliteit is hoog en ook de hoeveelheid valt niet tegen. Het meest spectaculaire is natuurlijk weer TCalc, de spreadsheet die Borland bij al haar talen meelevert. De voorbeelden van TurboVision zijn goed en geven een uitstekend beeld van wat met dit framework te bereiken is.

Helpsysteem

Het helpsysteem van TP geeft geen reden tot ontevredenheid. De rechtermuisknop geeft context-gevoelige hulp, terwijl het gevraagde onderwerp ook op andere manieren makkelijk is op te zoeken. Bij elke standaard functie wordt een voorbeeld gegeven, zodat de manier waarop zo'n functie gebruikt dient te worden meestal wel duidelijk wordt.

TurboVision

Veruit het interessantste aan TP is het meegeleverde framework TurboVision (TV). Elders in deze DOS-Special is hier een artikel aan gewijd, zodat we er hier niet al te diep op in zullen gaan. TV maakt het mogelijk om SAA-compatible applicaties te schrijven, zonder dat u hier iedere keer 'from scratch' mee hoeft te beginnen. Het framework is uiteraard object-georiënteerd.

Conclusie

Het is duidelijk, Borland wint met Turbo Pascal op punten. De meegeleverde documentatie is stukken beter dan die van Microsoft en ook het helpsysteem heeft bij TP meer aandacht gekregen. QP biedt u slechts een matige compiler, gecompileerde code was steevast groter, en geeft u slechts een aardige IDE. Programmeurs die zich op OOP gaan richten dienen al helemaal hun toevlucht te zoeken tot Borland. QuickPascal is aardig om Pascal mee te leren, maar om een professionele applicatie mee te maken is het niet geschikt.

Scramble in QuickBasic en assembly

In de vorige DOS-Special schitterde de programmeertaal BASIC door afwezigheid. Nu wordt er regelmatig afgegeven op Basic als ware het een 'laag-bij-de-grondse' ('basic') taal waar niets serieus mee te doen valt. Desondanks bespeur je soms toch wat afgunst, zoals in de boekbespreking in die vorige DOS-Special van 'C voor BASIC-programmeurs' (Pim Philipse) waarin de recensent peinst dat dit boek 'ook is voor hen die soms met weemoed terugdenken aan het gemak waarmee in BASIC vele ingewikkelde operaties met een enkel statement uitgevoerd konden worden...'.

Soms vraag je je af waarom men de moeite neemt om naar C over te schakelen. Basic is een zeer krachtige taal, die tegenwoordig ook een interface heeft naar de DOS en BIOS interrupts. Basic is leesbaar, begrijpelijk, doorzichtig, beknopt en is eigenlijk alleen door het idee van 'gestruktureerd programmeren' in een tijd dat Basic programma's nog vol met GOTO's stonden uit de gratie geraakt. Iedere taal kent overigens GOTO in enigerlei vorm, en mits met beleid gebruikt valt met GOTO soms heel veel te bereiken, zoals in de volgende 'Pseudo'code voor het testen en na goedkeuring openen van een bestand.

Zonder GOTO:

```
DO
ProbeerOpnieuw = false
IF conditie THEN
HerstelBestand
TestOpnieuw
ProbeerOpnieuw = true
ELSE
OpenBestand
ENDIF
LOOP WHILE ProbeerOpnieuw

Hetzelfde effect met GOTO:

Label1:
IF conditie THEN
HerstelBestand
TestOpnieuw
GOTO Label1
ELSE
OpenBestand
ENDIF
```

Een gefingeerd voorbeeld als dit laat al wat besparingen zien: een variabele minder, een vergelijking aan het eind van de LOOP minder, minder inspringen en dus leesbaardere code. Als tussen de IF ... ELSE ook nog drie geneste IF's of LOOP's zitten, weet u al waar de fouten gaan komen! Daarbij, GOTO zit zelfs in de meest gestruktureerde programmeertaal, want wat is 'IF conditie THEN .... ELSE' anders dan 'IF NOT conditie GOTO ELSE'?

Met Basic vallen zeer serieuze applicaties te bouwen, indien je de grootte van het uiteindelijke EXE-bestand op de koop toe neemt. Mocht dat een overweging zijn om naar een andere taal uit te wijken dan raad ik aan eens rond te kijken. Er bestaat een alternatieve Link-library waarmee EXE-bestand tot de helft kunnen slinken en dus ook sneller worden. Verder kan er met deze library een TSR van 1 of 2 kB in Basic geschreven worden. Met de QuickBasic compiler is niks mis, maar het uiteindelijke EXE-bestand is wat groot omdat er wat veel ongebruikte routines worden mee-gelinkt, hetgeen lang niet altijd nodig is.

Zo nu en dan moet men ook in Basic toevlucht nemen tot assembly, net als in andere talen. Dit geldt met name voor zaken die echt snel moeten zijn, zoals schermuitvoer, strings doorzoeken en dergelijke. Zodra de gebruiker moet wachten, wordt een programma als 'langzaam', en dus als onprettig ervaren.

Assembly gebruiken in Basic is heel simpel: Je schrijft een routine in assembly en maakt daarvan een QuickLibrary met behulp van de bij QuickBasic meegeleverde programma's. Vervolgens start men QuickBasic met deze library, waarna de routine getest en gebruikt kan worden. Zoals met iedere taal geldt ook hier dat er wel geweten moet worden hoe de taal in elkaar steekt. Zo mag men vanuit assembly alles met Basic strings

doen, zolang de plaats en de lengte maar niet veranderd worden. Het eerste, simpele, voorbeeld dat een getal met twee vermenigvuldigt illustreert de principes daarvan. De code is geschreven voor A86, een ShareWare assembler (verkrijgbaar via ToneHalt), maar is eenvoudig voor MASM of TASM aan te passen. De listing staat op de volgende pagina.

Een andere reden om naar assembly uit te wijken is bit-operaties. In de vorige DOS-Special (2/91) stond een versleutelings-routine voor Clipper waarmee bepaalde velden gecodeerd konden worden. De tweede listing in dit artikel geeft er een die in assembly geschreven is en vanuit Basic is aan te roepen (met een enkele wijziging ook vanuit C of Pascal.)

Het principe is simpel. We nemen aan dat versleuteling plaatsvindt vlak vóór het schrijven naar disk en direkt ná het lezen van disk. Versleutelen is namelijk alleen van belang voor zaken die op disk staan.

We gaan uit van binaire bestanden, dat wil zeggen dat je op iedere gewenste offset vanaf het begin van het bestand kunt gaan lezen of schrijven, en iedere gewenste lengte kunt lezen en schrijven.

Voor het algoritme gelden de volgende overwegingen:

- Gekozen is voor XOR-operaties, zodat zowel encrypt als decrypt met dezelfde code gedaan kunnen worden.
- De versleutelde data op disk moeten vanaf iedere positie weer 'ontsleuteld' kunnen worden. Als ik 'Hallo wereld' versleutel en op positie 1 in het bestand wegschrijf, vervolgens vanaf positie 7 iets lees en 'ontsleutel', moet ik 'wereld' krijgen!
- Het wachtwoord moet onleesbaar en on-ontcijferbaar blijven (en zichzelf ook niet herhalen) als ik een lang stuk spaties of een lang stuk nul-karakters versleutel.
- Een 'leeg' wachtwoord moet geen enkele versleuteling veroorzaken.

Het algoritme is verder simpel gehouden om snelheid te verkrijgen. Een geoefende cryptoloog zal versleutelde bestanden vast wel kunnen ontcijferen, maar zolang een leek denkt dat het om een programma-bestand gaat als deze versleutelde data ziet is ons doel bereikt.

In de komende kaders staan de assembly code, de link opdrachten, en BASIC voorbeeld programmaatjes, inclusief uitleg.

Dennis ten Siethoff

```
;Getal:   16384, resultaat  32767, geen teken
;Call:      b% = Twice(a%)
;Declare:   DECLARE FUNCTION Twice(BYVAL a%)

Twice:                  ;naam van de funktie
      push bp           ;bewaar bp
mov bp,sp;gebruik bp nu om de
                        ;stack te adresseren,
                        ;want daar staat a%

      mov ax,[bp+6]     ;zet a% in ax
      clc               ;clear carry, anders
                        ;wordt die wellicht
                        ;ax binnengeschoven...
shl ax,1;vermenigvuldig met 2
                        ;laat het resultaat
                        ;in AX staan

      mov sp,bp         ;herstel de stack
                        ;pointer, zodat we
                        ;weten waar ons
                        ;terugkeer-adres
                        ;naar BASIC staat
      pop bp            ;herstel bp
      retf 2            ;keer terug
;Einde van de routine

;Maak hiervan een OBJ-file vanaf de command-line
;met bijvoorbeeld A86:
;  C:   A86 twice.asm TO twice.obj
;Maak een quick-library:
;  C:  Link /q twice.obj, twice.qlb, , bqlb41.lib
;Start QuickBasic:
;  C:  QB /l twice
;Schrijf een test-programma:
;
;DEFINT A-Z
;DECLARE FUNCTION Twice(BYVAL a%)
;
;CLS : FOR a = 0 TO 16384: PRINT Twice(a); : NEXT
;END
;
;De tafel van twee...
```

```
;————————————————————————;
quick (un)scrambling algorithm
;
;Call:          Scramble text$, password$,
offset&;
;               See QB example program
;
;Return:        (un)scrambled text$
;
;Author:        D. ten Siethoff 1991
;
;————————————————————————;
Scramble:
     a$     = [bp+12]   ;text to be (un)scrambled
     b$     = [bp+10]   ;password$ (any length)
     offshi = [bp+8]    ;longint = file offset&
     offslo = [bp+6]

     push bp                    ;entry code
     mov bp,sp
```

```
     push si,di              ;save registers used

;Calculate: bx = offset& MOD LEN(PassWord$)
     mov dx,offshi      ;dx:ax = offs&
     mov ax,offslo
     mov di,b$          ;get b$, password
     mov bx,[di]        ;length of b$
     xor bh,bh          ;use up to 255 bytes
                                  ;of the
password
          cmp bl,0                ;password
empty?
     je L2             ;don't scramble!
     mov di,[di+2]      ;address of b$
                                      ;di:bl =
password$/b$
     div bx             ;divide dx:ax by bx, get

;remainder
in dx
     xchg bx,dx         ;bx = current offset into

;password$
                        ;dl = length password$

;Calculate: dh = seed rotated (offset& mod 8)
times
     mov cx,offslo
     and cx,7           ;cx = longint mod 8
     mov dh,137         ;fixed seed 10001001, any
                                   ;seed will
do, but an
                        ;a-symmetric pattern works
                                       ;best
     ror dh,cl          ;dh = scramble byte

     mov si,a$          ;get a$, text
     mov cx,[si]
     jcxz L2           ;no text? Then go home!
     mov si,[si+2]     ;si:cx is address/len of a$

;Scramble away...
L0:  lodsb              ;get char from a$
     xor al,[di+bx]     ;(un)scramble with
password
     xor al,dh          ;(un)scramble with
position
     mov [si-1],al      ;store (un)scrambled char.

     ror dh,1           ;next position in file
     inc bx             ;next char in password
     cmp bl,dl          ;beyond length of
password?
                        ;(zero based!)
     if ae xor bx,bx    ;then zero again
L1:  loop L0

;Tidy up...
L2:  pop di,si            ;restore registers
     mov sp,bp            ;restore stack
     pop bp
     retf 8               ;return skip 4 args.

 ;————————————————————————;
;Notities: De string wordt versleuteld met
telkens
;een ander karakter van het wachtwoord, en met
een
;byte die verbonden is met de positie in het be-
;stand waar de string wordt geschreven. Aangezien
;deze laatste byte iedere 8 posities hetzelfde
is,
;is het aan te raden om een wachtwoord te kiezen
;met een oneven aantal karakters. Bij 7 karakters
;wordt een string van spaties of nul-karakters
elke
;56 karakters hetzelfde versleuteld.
;Verder is het aan te raden om in het programma
;zelf een wachtwoord te voorzien van een 'stan-
;daard'-uitbreiding, zodat een wachtwoord van n
;letter toch tot een redelijke versleuteling
leidt.
;B.v.: PassWord$ = PassWord$ + "Scramble".
;————————————————————————;

 ;————————————————————————;
;Compileer:
;Maak een OBJ-file vanaf de command-line met bij-
;voorbeeld A86:
;A86 scramble.asm TO scramble.obj
;Maak een quick-library:
;Link /q scramble.obj, scramble.qlb, , bqlb41.lib
;Start QuickBasic:
;QB /l scramble
;————————————————————————;

 ;————————————————————————;
'SCRAMBLE.BAS - Scramble test-programma
DEFINT A-Z
DECLARE SUB Scramble (text$, password$, BYVAL
                                       offs&)

OPEN "Scramble.tst" FOR BINARY AS 1
a$ = "Dit is een test" + SPACE$(1000)
LINE INPUT "Password: "; pw$
IF LEN(pw$) THEN pw$ = pw$ + "Scramble"
'Empty password$? Then don't alter a$!!!
Scramble a$, pw$, 1       'scramble
PUT 1, 1, a$                      'write to file

a$ = space$(1015)
GET 1, 1, a$                      'read from file
Scramble a$, pw$, 1       'unscramble
PRINT "Unscrambled should read: Dit is een test"
PRINT LEFT$(a$, 80)

a$=space$(1003)
GET 1, 12, a$                     'read from another
position
Scramble a$, pw$, 12      'unscramble
PRINT "Unscrambled should read: test"
PRINT LEFT$(a$, 80)

PRINT "Now SHELL to DOS, and use LIST.COM to "
PRINT "inspect Scramble.TST which should be "
PRINT "illegible!!"
END

;————————————————————————;
NB: Als je een versleuteld bestand afsluit is
het natuurlijk mooi als de sleutel zlf ook 'ver-
sleuteld' in dat bestand staat. Dan kun je bij het
openen van een bestand direkt naar het wachtwoord
vragen en het bestand niet openen als dat wacht-
woord fout is. Het simpelste is om het wachtwoord
te versleutelen met een vaste code, en het aan het
eind van het bestand te zetten. Dit doen we wel zo
dat het wachtwoord in een string met een vaste
lengte staat, zodat het onduidelijk blijft hoelang
het wachtwoord zelf is:

a$ = SPACE$(255): LSET a$ = pw$
Scramble a$, "Versleutel", LOF(1)
PUT 1, LOF(1), a$: CLOSE 1

Na openen heb je zo het wachtwoord:
a$ = SPACE$ (255)              'moet genoeg
zijn
offs& = LOF(1) - 255 + 1       'waar staat het?
GET 1, offs&, a$               'string ophalen
Scramble a$, "Versleutel", offs& 'leesbaar maken
eind = INSTR(a$, "Scramble")
pw$ = LEFT$(a$, eind - 1)      'haal "Scramble" eraf

'Kontroleer of gebruiker het juiste woord weet:
LINE INPUT "Geef wachtwoord: ";a$
if a$  pw$ THEN PRINT "Sorry, foutje!": END
'Rest programma
;————————————————————————;
```

Actor 3.0

Actor is een uitgebreide ontwikkelingsomgeving en programmeertaal voor het ontwikkelen van stand-alone Windows applicaties. Deze in 1987 door The Whitewater Group geïntroduceerde taal laat een duidelijk andere aanpak zien van hoe een Windows-applicatie tot stand kan komen. Zo is Actor in tegenstelling tot bijvoorbeeld Borland C++ of Turbo Pascal for Windows een puur object-georiënteerd systeem: alles is een object. De omgeving bevat bovendien een incremental compiler, wat tot gevolg heeft dat wijzigingen in een programma niet leiden tot volledige hercompilatie, maar direkt worden verwerkt.

OOP en Actor

Objecten zijn de bouwstenen van een object-georiënteerd programma en spelen dan ook een centrale rol in een taal als Actor. Ze hebben gedetailleerde kennis van de taken die ze moeten verrichten; zulke taken hebben meestal betrekking op een bepaald onderwerp. Er zijn in de Actor omgeving meer dan 125 objecten gedefinieerd waar men al direct mee aan de slag kan. Zo kent de taal een object dat alles weet van popup windows. Aan dit object kan men berichten sturen zoals 'open jezelf' of 'verplaats jezelf', enzovoort.

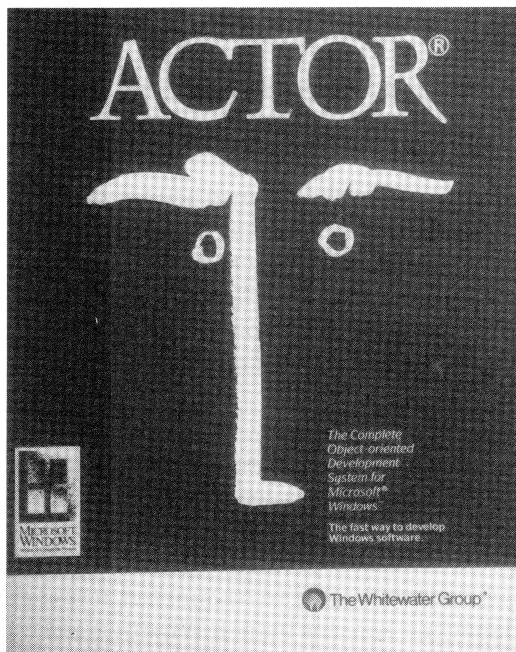
Een object wordt gedefinieerd in een class. Deze class bevat zowel de implementaties van de berichten die je het object kunt sturen als de data die het object kan bevatten. De implementatie van een bericht wordt in een object-georiënteerde omgeving aangeduid met methode en is eigenlijk niets anders dan een functiedefinitie zoals men die ook wel in een taal als C of Pascal tegenkomt. De algemene syntaxis voor het verzenden van een bericht in Actor is:

```
Bericht(Ontvanger, Argument,
Argument...);
```

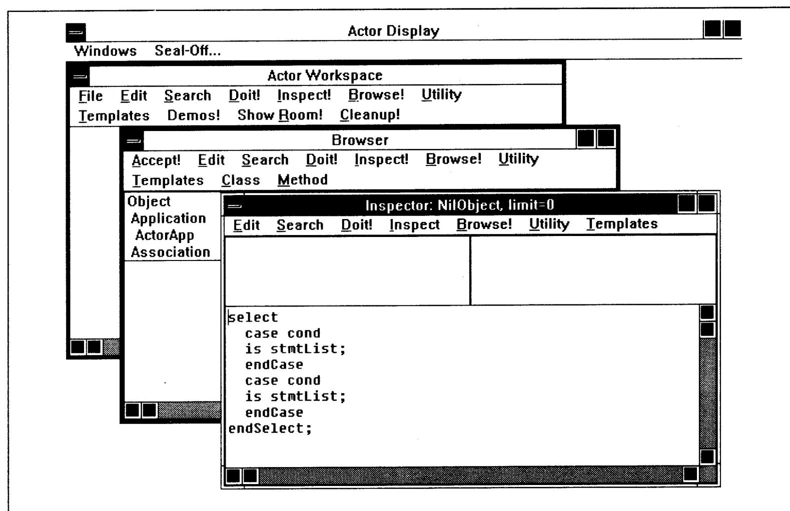
waarbij Ontvanger staat voor het object waarna het bericht verzonden wordt. Toch is het verzenden van een bericht in Actor algemener dan een functie aanroep in C of Pascal; verschillende objecten kunnen ieder op hun eigen manier op zo 'n bericht reagereren:

```
show(EditWindow, 3);
show(AboutWindow, 1);
```

Het sturen van dezelfde berichten naar verschillende objecten staat bekend als polymorfisme en maakt het programmeren een stuk overzichtelijker. De ontvanger van een bericht reageert door zijn overeenkomstige methode te raadplegen.



Inheritance of overerving is de techniek dat bestaande objecten gebruikt worden als basis voor nieuwe objecten. Deze nieuwe objecten erven de eigenschappen van de basis-objecten en kunnen dus als uitbreiding op hun 'ouders' worden gezien. Hierdoor gaat de kennis die in het basis-object ligt opgeslagen niet verloren. Zo zijn de classes van EditWindow en AboutWindow gedefinieerd met een gemeenschappelijke basis-class: Window. Inheritance kan gebruikt worden om



Actor ontwikkelomgeving

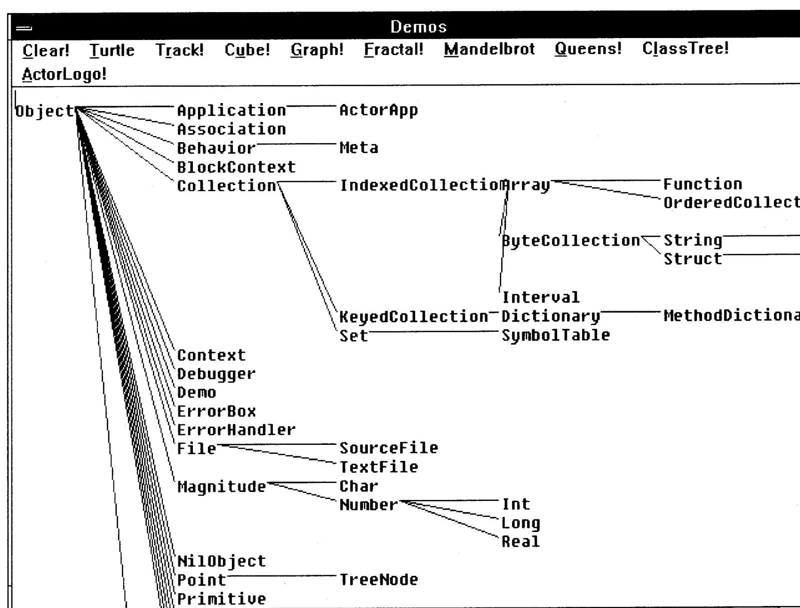
een hiërarchisiche boomstructuur van gerelateerde classes te bouwen, waarbij de bovenliggende classes de eigenschappen definiëren die voor alle onderliggende classes gelden, zodat deze niet voor de afgeleide classes opnieuw gedefinieerd hoeven worden.

De Actor compiler wordt standaard geleverd met een aantal tools zoals browsers, object inspectors en debuggers die alle als popup windows geactiveerd kunnen worden. Het gehele proces van programmeren, testen en debuggen kan dus binnen Windows zelf plaats vinden. Het belangrijkste window in de Actor omgeving is de Workspace; vanuit dit window worden onder andere de source files geladen en kunnen de diverse tools geactiveerd worden. De primaire functie van de Workspace is het testen van de classes en methoden die in de Browsers gecreëerd zijn. Dit testen gebeurt met de ingebouwde command processor; je voert een regel code in en executeert hem. Zo kun je gebruikmakend van de al in Actor gedefinieerde class EditWindow al een eenvoudige Editor op het scherm zetten:

```
SimpelEdit := defaultNew(EditWindow,
"Simpele Editor");
show(SimpelEdit, 1);
```

De eerste regel creëert een object of instantie van class EditWindow met als titel 'Simpele Editor' en de tweede regel laat de zojuist gecreëerde instantie van EditWindow op het scherm zien. Aangezien 'SimpelEdit' vanaf dit moment als applicatie onder Actor actief is kan deze ook werkelijk als editor gebruikt worden en kun je een optie uit zijn systeem menu activeren. De command processor maakt het dus mogelijk om op eenvoudige wijze snel een toevoeging of wijziging in de source code te testen.

Zodra de testfase achter de rug is en de applicatie naar behoren onder Actor werkt, kan men overgaan tot het creëeren van een stand-alone, gecompileerde versie voor de eindgebruikers. Het creëeren van een stand-alone applicatie, staat onder Actor bekend als 'sealing off' applicatie en is een proces dat zich duidelijk onderscheidt van andere programmeer-omgevingen. Een Actor applicatie bestaat namelijk uit twee delen, een image file (.IMA) dat het gecompileerde programma bevat, en een executeerbare file (.EXE) met de kernel van het Actor systeem en de Windows resources waar het programma gebruik van maakt. Omdat alles in Actor gezien wordt als een object is het niet verwonderlijk dat de applicatie zelf ook een object is. Alle Actor applicaties moeten een class definiëren waarbinnen een main object gecreëerd wordt. Deze class staat bekend als de applicatie class en moet een afgeleide zijn van de in Actor gedefinieerde 'Application class'.

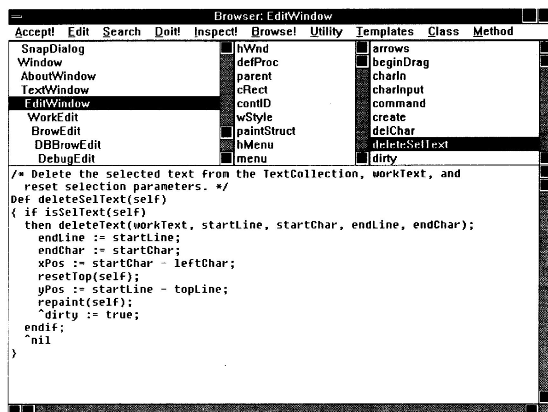


Voorgedefinieerde objecten

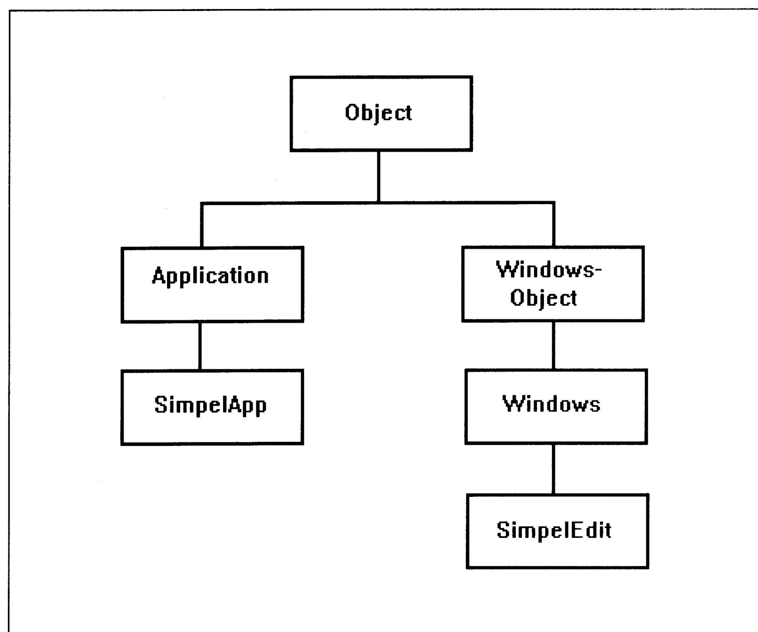
Als we van SimpelEdit uit het voorbeeld hierboven een stand-alone, Windows 3.0 applicatie willen maken zullen we dus een afgeleide class van Application moeten definiëren waarin een instantie van EditWindow wordt gecreëerd. Stel dat we deze afgeleide class SimpelApp noemen, dan moet SimpelApp tenminste een eigen init() methode bevatten om de applicatie op te kunnen starten:

```
Def init(self, commandStr)
{
  init(self:ancestor, commandStr);
  mainWindow := newMain(EditWindow, nil,
"Simpele Editor", nil);
  show(mainWindow, CmdShow);
}
```

De eerste regel verzendt een init() bericht naar de 'ouder' van SimpelApp (class Application). Deze verricht belangrijke systeeminitialisaties die het mogelijk maken dat ook andere Windows-programma's gedraaid kunnen worden. De volgende regel creëert SimpelApp's main window als een 'overlapped window' en kent dit object toe aan de in Application class gedeclareerde variabele mainWindow. Tenslotte wordt er een show() bericht verzonden naar het main window object zodat deze op het scherm verschijnt. De CmdShow variabele is globaal gedeclareerd en specificeerd het formaat waarin het window geopend dient te worden (open, iconic, etc.). Wanneer de gebruiker van SimpelEdit de sluiten-optie uit het menu kiest, zal Windows een shouldClose() bericht naar de applicatie sturen. Stel dat onze applicatie gebruik zou maken van bestanden dan zouden we dit bericht af kunnen vangen met een gelijknamige methode:



Object Browser



SimpleApp's objecten-structuur

```
Def shouldClose(self)
{
  if shouldClose(self:ancestor) then
    bewaar bestanden...
    sluit bestanden af...
  else ^nil;
  endif;
}
```

Nu het programmeerwerk achter de rug is, kunnen we overgaan tot het creëeren van de .IMA file. Door de 'seal-off' optie uit het Actor menu te kiezen wordt er een proces in werking gezet dat alleen die classes, methoden en objecten aan het image file toevoegd welke nodig zijn voor de executie van de uiteindelijke applicatie. Op de wat minder snelle machines kan dit proces enige minuten in beslag nemen. De uitgebreide test mogelijkheden binnen Actor zelf zijn dan ook geen overbodige luxe. De laatste stap is het creëren van de executeerbare (.EXE) file die de image moet gaan draaien en de resources bevat. Deze resources worden gedefinieerd in resource file (.RC) en uiteindelijk aan de kernel (ACTOR.EXE) toegevoegd.

```
COPY ACTOR.EXE SIMPELAP.EXE
COPY USERAPP.RC SIMPELAP.RC
RC SIMPELAP
WIN SIMPELAP
```

Wat overblijft is SIMPELAP.EXE en SIMPELAP.IMA; twee files van een aanzienlijke omvang. Het bij de Actor meegeleverde ontwikkelgereedschap is zeer functioneel. Zo heeft

de omgeving een gespecialiseerde editor (browser) voor het wijzigen van al bestaande of het implementeren van nieuwe classes. Deze tool kan als popup window vanuit de Workspace geactiveerd worden. Iedere wijziging van de code in een Browser wordt automatisch gecompileerd en opgeslagen op disk zodat eventuele fouten in een vroeg stadium aan het licht komen.

De Browser is opgedeeld in totaal vier windows. Een class window waarin een overzicht van de al bestaande classes. Een variabelen window met de gedefinieerde variabelen uit de geselecteerde class en zijn basis-classes. Een methode window met de object methoden van de class en een edit window met de implementatie van de geselecteerde methode.

Wanneer de Actor omgeving een run-time error in de code detecteert wordt automatisch een Dialog Window geopend. Een dergelijke window geeft niet alleen de foutmelding maar tevens een overzicht van de programma-executie in de vorm van methode-aanroepen zodat de locatie en oorzaak van de fout eenvoudiger te achterhalen is. Na het fixen van de bug is het mogelijk de executie van het programma vlak voor het punt dat de run-time error gedetecteerd werd te hervatten. Ondanks het grote assortiment aan classes en de functionele tools die bij Actor meegeleverd worden heeft deze omgeving toch een paar nadelen. Allereerst moet men zich een nieuwe taal eigen maken; in de programmeertalen Pascal en C/C++, eventueel aangevuld met class libraries van derden, kan men immers ook Windows programmeren. Verder zorgt het executeerbare eindprodukt voor een behoorlijke aantasting van de schijfruimte.

Non-stop ASCII

Selkey: zeg vaarwel tegen de ASCII-tabel

Selkey is een Memory Resident programma waarmee u met één druk op een toets elke ASCII-code geselecteerd en ingevoerd kan worden in het keyboard-buffer. Hierdoor is het omslachtige "NumLock + Alt + ASCII-code" definitief verleden tijd.

```
        01234567891123456789212345678931234567894123456789
  0 -    ☺☻♥♦♣♠•◘○◙♂♀♪♫☼►◄↕‼¶§▬↨↑↓→←∟↔▲▼
100 -                              ⌂ÇüéâäàåçêëèïîìÄÅÉæÆôöò
150 -   ûùÿÖÜ¢£¥₧ƒáíóúñÑªº¿⌐¬½¼¡«»░▒▓│┤╡╢╖╕╣║╗╝╜╛┐└┴┬├─┼╞╟
200 -   ╚╔╩╦╠═╬╧╨╤╥╙╘╒╓╫╪┘┌█▄▌▐▀αßΓπΣσµτΦΘΩδ∞φε∩≡±≥≤⌠⌡÷≈°∙
250 -   ·√ⁿ²■
        01234567891123456789212345678931234567894123456789
```

Selkey moet geladen worden vóórdat U het hoofdprogramma laadt waarin u de ASCII-codes wilt gaan gebruiken. Na indrukken van de Hot-Key (in dit geval Alt + 5 op num-pad) verschijnt een ASCII-tabel op het scherm en kan met de cursortoetsen het gewenste teken geselecteerd worden. Na een druk op return wordt dit teken in de keyboard-buffer gezet en zo overgedragen aan het hoofd-programma. Een getal zal een overeenkomstig aantal toetsen invoeren (een nul 10 maal). Na escape wordt Selkey verlaten zonder een teken in te voeren.

WAARSCHUWING: Er is een aantal programma's, die hun eigen keyboard routines installeren. Dit kan problemen opleveren en mogelijk een systeem crash veroorzaken. Met bijvoorbeeld WP 4.2 werkt dit programma niet. Bij met Turbo Pascal 5.5 en met TP 5.5 geschreven programma's zijn geen problemen opgetreden.

Opmerking: Dit programma kan ook dienen als basis voor zelf te schrijven TSR-programma's. Alleen de procedure User_Routine en de in deze procedure gebruikte variabelen dienen vervangen te worden, hetgeen in de listing is aangegeven.

H.M. Leeflang

```
{$A-,B-,D-,E-,F+,I-,L-,N-,O-,R-,S-,V+}
{$M 2000, 200, 200}

PROGRAM Selkey;

{
   (C) 1991 H.M. Leeflang

  Adres: Terrastraat 26
         1829 XM Oudorp
  Taal:  Turbo Pascal
  Versie: 5.5 }

USES Crt, DOS;
{ Alle variabelen zijn globale variabelen omdat
  dan niet bij elke oproep van het TSR-programma
  opnieuw ruimte in het geheugen gereserveerd
  hoeft te worden. }
{ Variabelen t.b.v. het TSR programma. }
CONST
 Programma_Naam: STRING[8] = 'SELKEY';
{ Vul hier de naam van Uw TSR programma in      }
 ScanCode = 76;                 { 5 op num-pad   }
 KeyMask = $08;                 { Alt      = $08 }
                                { Ctrl     = $04 }
                                { Sh left  = $02 }
                                { Sh right = $01 }
VAR
 KbdIntVec: PROCEDURE ;        { De oude key-board }
                               { interrupt         }
 TsrBezig: Boolean;

{ Variabelen t.b.v. User_Routine }
CONST
 Normal = Black SHL 4 + White; {Zelf te wijzigen}
 Invers = LightGray SHL 4 + Black;     { Idem }
 Hex: STRING[16] = '0123456789ABCDEF';

TYPE
 Scherm_Ptr = ^Scherm_Array;
 Scherm_Array = ARRAY [1..19, 1..80] OF Word;
{Aangezien het programma alleen regels 1 t/m 19
 gebruikt worden alleen deze regels opgeslagen.
 Indien wel het hele scherm gebruikt wordt dient
 hier 25 (of 53 i.g.v. VGA/EGA scherm in 53-regel
```

```
 mode) ingevuld te worden. }

VAR
 Scherm_Pointer: Scherm_Ptr;{ Wijst naar actueel}
                            { video geheugen     }
 Bewaar:                    { Bewaren scherm     }
  RECORD                    { informatie         }
   Scherm: Scherm_Array;
   X, Y, Attr: Byte;
  END;
 Klaar: Boolean;
 R: Registers;
 Kbd_Next_Key: Byte ABSOLUTE $0000: $041C;
 Kbd_Buffer: ARRAY [0..15] OF RECORD
   Asc, Scan: Byte
  END ABSOLUTE $0000: $041E;
{ De bovenstaande variabelen worden gebruikt om
  een toets in de keyboard-buffer op te slaan.
  Zie de procedure Fop_Kbd_Buffer. }
 Huidig_Kar, Vorig_Kar, Key, B: Byte;

{$F+} { Zeer belangrijk }

PROCEDURE User_Routine;
{ Vervang eventueel deze procedure door een zelf
  geschreven procedure. }

 PROCEDURE Fop_Kbd_Buffer(Kar, Aantal: Byte);
{ In deze procedure wordt het teken kar op de
  volgende positie in de buffer gezet. Aantal
  geeft aan hoe vaak dit dient te gebeuren.
}
  BEGIN
   WHILE Aantal > 0 DO
    BEGIN
     Kbd_Buffer[Kbd_Next_Key DIV 2 - 15].Asc :=
      Kar;
     Inc(Kbd_Next_Key, 2);
     IF Kbd_Next_Key > 60 THEN
      Kbd_Next_Key := 30;
     Dec(Aantal);
    END;
  END;

 PROCEDURE Print_Op(X, Y, A: Byte;
                    S: STRING);
{ Deze procedure print een tekst-string op positie
  (X, Y) op het scherm in kleur A.
}
  BEGIN
   GotoXy(X, Y);
   TextAttr := A;
   Write(S);
  END;

 PROCEDURE Print_Char(Kar, Attr: Byte);
{ De standaard procedure Write toont niet alle
  ASCII codes onder de 32. Deze procedure toont
  het teken Kar op de positie van de cursor met
  attribute Attr.
}
  BEGIN
   WITH R DO
   BEGIN
    AH := 9;
    AL := Kar;
    BH := 0;
    CX := 1;
    BL := Attr;
    Intr($10, R);
   END;
  END;

 PROCEDURE Print_Cursor(X, A: Byte);
{ Deze procedure maakt een cursor, bestaande uit
  een Space, het teken en een Space. De kleur
  wordt bepaald door het byte A.
}
  BEGIN
   Print_Op((X MOD 16) * 3 + 2,
          (X DIV 16) + 2, A, #32);
   Print_Char(X, A);
   Print_Op(Succ(WhereX), WhereY, A, #32);
  END;

 PROCEDURE Maak_Scherm;
{ Deze procedure wist eerst een deel van het
  scherm en toont dan de ASCII-tabel.
}
  BEGIN
   WITH R DO
    BEGIN
     AX := 6 SHL 8;
     CX := 0;
     DX := 18 SHL 8 + 50;
     BH := Normal;
{ Wis alleen regels 1 t/m 19 en Kollom 1 t/m 50.}
     Intr($10, R);
    END;
   FOR B := 1 TO 16 DO
    BEGIN
     Print_Op(Pred(B * 3), 1, Normal, ' ' +
              Hex[B] + ' ');
     Print_Op(1, Succ(B), Normal, Hex[B]);
    END;
   FOR B := 0 TO 255 DO
    Print_Cursor(B, Normal);
  END;

 PROCEDURE Toon_Informatie;
{ Deze procedure toont eerst een inverse cursor op
  het huidige karakter en wist de inverse cursor
  op het vorige karakter. Bovendien wordt een
  informatie regel met de ASCII code en het
  huidige karakter getoond.
}
  BEGIN
   Print_Cursor(Huidig_Kar, Invers);
   IF Huidig_Kar <> Vorig_Kar THEN
    Print_Cursor(Vorig_Kar, Normal);
   Print_Op(1, 19, Normal, 'ASCII code: ');
   Write(Huidig_Kar: 3, ' ($' +
         Hex[Succ(Huidig_Kar SHR 4)] +
         Hex[Succ(Huidig_Kar MOD 16)] +
         ')  Char: ');
   Print_Char(Huidig_Kar, Normal);
   Vorig_Kar := Huidig_Kar;
  END;
 BEGIN
  WITH Bewaar DO
   BEGIN
    X := WhereX;
    Y := WhereY;
    Attr := TextAttr;
    Scherm := Scherm_Pointer^;
   END;
{ Bewaar de informatie, die oorspronkelijk op het
  scherm stond.}
  Maak_Scherm;
  Klaar := False;
  REPEAT
   Toon_Informatie;
   Key := Ord(Readkey);
   CASE Key OF
    27: Klaar := True;
    13: BEGIN
         Fop_Kbd_Buffer(Huidig_Kar, 1);
         Klaar := True;
        END;
    48..57: BEGIN
             IF Key = 48 THEN Key := 10
             ELSE Key := Key - 48;
             Fop_Kbd_Buffer(Huidig_Kar, key);
             Klaar := True;
            END;
    0: CASE Readkey OF
        #75: Dec(Huidig_Kar);      {Left}
        #77: Inc(Huidig_Kar);      {Right}
        #72: Dec(Huidig_Kar, 16);  {Up}
        #80: Inc(Huidig_Kar, 16);  {Down}
```

```
        #71: dec(Huidig_Kar, 17); {Home}
        #79: Inc(Huidig_Kar, 15); {End}
        #73: Dec(Huidig_Kar, 15); {PgUp}
        #81: Inc(Huidig_Kar, 17); {PgDn}
       END;
   END;
  UNTIL Klaar;
  WITH Bewaar DO
   BEGIN
    Scherm_Pointer^ := Scherm;
    GotoXy(X, Y);
    TextAttr := Attr;
   END;
{Herstel de oude scherm-informatie.}
 END; {User_Routine}
{ De nieuwe Keyboard interrupt. }

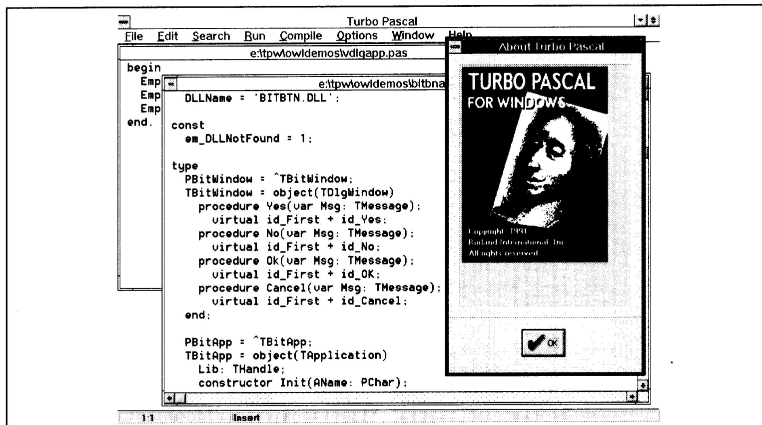
PROCEDURE NewKbdInt;INTERRUPT;
{Deze procedure komt in de plaats van de stan-
 daard keyboard interrupt. Indien de Hot-Key
 (gedefinierd door ScanCode en KeyMask) inge-
 drukt wordt zal het programma User_Routine
 opgeroepen worden. In alle andere gevallen wordt
 de oude keyboard interrupt uitgevoerd.
}
 BEGIN
  IF NOT TsrBezig AND
     (Port[$60] = ScanCode) AND
     ((Mem[$0000: $0417] AND
     KeyMask) = KeyMask) THEN
   BEGIN
    B := Port[$61];
    Port[$61] := B OR $80;
    Port[$61] := B;
{   Reset het toetsenbord }
    INLINE ($FA); { CLI }
    Port[$20] := $20;
    INLINE ($FB); { STI }
    TsrBezig := True;
    INLINE ($FA);
    User_Routine;
    INLINE ($FB);
    TsrBezig := False;
    Exit;
   END
  ELSE
   BEGIN
    INLINE ($9C);
    KbdIntVec;
    Exit;
   END;
 END;
{$F-}

FUNCTION Al_Geladen(Huidig_Ident: Pointer):
          Boolean;
{Deze functie test of SelKey al geladen is. Dit
 gebeurt door de eerste keer een vrije interrupt
 adres naar de identificatie string te laten
 wijzen. Elke keer, dat het programma opnieuw
 geladen wordt, zal gekeken worden of er een
 interrupt adres gevuld is met een adres en of de
 string op dit adres gelijk is aan de identifica-
 tie string.
}
 VAR
  Vorig_Ident: Pointer;
  Vector: Word;
 BEGIN
  Al_Geladen := False;
  FOR Vector := $60 TO $67 DO
{ Dit zijn de vrije interrupts. }
   BEGIN
    GetIntVec(Vector, Vorig_Ident);
    IF Vorig_Ident = NIL THEN
     BEGIN
      SetIntVec(Vector, Huidig_Ident);
      Exit;
     END;
    IF STRING(Vorig_Ident^) =
       STRING(Huidig_Ident^) THEN
     BEGIN
      Al_Geladen := True;
      Exit;
     END;
   END;
 END;

{ ** Begin van het programma. ** }
BEGIN
 IF Al_Geladen(@Programma_Naam) THEN
  WriteLn(Programma_Naam + ' al geladen.')
 ELSE
  BEGIN
   TextAttr := TextAttr + Blink;
   WriteLn(' * * * * * ' + Programma_Naam +
    ' * * * * *');
   TextAttr := TextAttr AND NOT Blink;
{ Vervang onderstaande regels bij een zelf ge-
  schreven programma.
  Laat Scherm_Pointer naar het begin van het
  videoscherm wijzen.}
   IF ((Mem[0000: 1040] AND 48) <> 48) THEN
    Scherm_Pointer := Ptr($B800, 0)
   ELSE Scherm_Pointer := Ptr($B000, 0);
   WriteLn('ASCII code selectie programma.');
   WriteLn('Met Alt 5 (NumPad) kunt U dit' +
    ' programma activeren.');
   WriteLn;
   Huidig_Kar := 0;
   Vorig_Kar := 0;
{ Bewaar de oude keyboard interrupt en vervang
  door nieuwe keyboard procedure.}
   GetIntVec($09, @KbdIntVec);
   SetIntVec($09, Addr(NewKbdInt));
   TsrBezig := False;
   Keep(0);
  END;
END.
```

Turbo Pascal voor Windows

Een tijdje geleden introduceerde Borland in het World Trade Centre te Amsterdam Turbo Pascal voor Windows. 'Voor Windows' kan hier op twee manieren worden uitgelegd. Turbo Pascal voor Windows (TPW) is namelijk een echte windows-applicatie, maar ook is het mogelijk om er windows-applicaties mee te schrijven. TPW wordt geleverd met ObjectWindows. Dit is een applicatie-framework dat u in staat stelt om op een eenvoudige manier windows-applicaties te ontwikkelen. Het gebruik van de SDK is hiervoor niet nodig!



Turbo Pascal for Windows

Het grote voordeel van een windows-hosted compiler als TPW is dat een applicatie geschreven, gedebugd, gecompileerd en gerund kan worden in Windows zelf. Daarbij levert Borland een framework mee dat het veel eenvoudiger maakt om Windows-applicaties te schrijven. Dit framework, ObjectWindows, geeft u vooraf gedefinieerde objecten die u alleen hoeft te instantiëren. Wie wel eens het overbekende 'Hello World' voor Windows heeft gezien weet hoe moeilijk het is om een Windows-applicatie te schrijven. Om dit in een window te toveren heeft u minstens honderd regels code nodig die zorg dragen voor het gedrag van zo'n window. Zo'n window moet opnieuw getekend worden indien een overlappend window gesloten wordt enzovoort. ObjectWindows neemt u al deze moeilijkheden uit handen. Door zaken als windows in objecten te plaatsen die u alleen hoeft te initialiseren hoeft u zich geen zorgen te maken of zo'n window wel opnieuw getekend wordt, die zorg lag bij de programmeurs van Borland. Een programma als 'Hello World' wordt dan vereenvoudigd tot de volgende code:

```
Program Hello;
Uses WObjects, WinTypes, WinProcs;

Type  THelloApp = object(TApplication)
        Procedure InitMainWindow;
virtual;
      End;

Procedure THelloApp.InitMainWindow;
Begin
  MainWindow := New(PWindow, Init(Nil,
'Hello World'));
End;

Var HelloApp: THelloApp;

Begin
  HelloApp.Init('HelloApp');
  HelloApp.Run;
  HelloApp.Done;
End.
```

Het window dat dit programma genereert heeft alle capaciteiten die een normaal window zou moeten hebben. U kunt het verslepen, vergroten en verkleinen, sluiten enzovoort. In feite is het een gewone windows-applicatie, omdat ze ook in de lijst voorkomt wanneer men op Ctrl-Esc drukt. Dit alles wordt verwezenlijkt in een programma van niet meer dan vijftien regels!

Objecthiërarchie

Het aantal objecten dat Borland meelevert in ObjectWindows is in totaal zo'n vijfentwintig stuks, waarvan u zelf weer eigen objecten kunt afleiden. Niet alle objecten zijn direct bruikbaar om instanties vanaf te leiden, maar worden gebruikt als een soort abstracte basisobjecten. Van TObject worden bijvoorbeeld TApplication, TWindowsObject en TScroller afgeleid, terwijl een instantie van TObject niet snel zal worden aangemaakt. TWindowsObject is dan weer een abstract object type dat als baseobject geldt voor TDialog en TWindow. TDialog op haar beurt is dan weer een baseobject voor TDlgWin-

dow, TFileDialog en TInputDialog. Het voordeel van zo'n abstract basisobject is natuurlijk hergebruik van code. Er wordt eerst een object gedefiniëerd dat de basisprincipes aangeeft, waarna een aantal afgeleide objecten specifieker gedefiniëerd worden.

Uses WinCrt;

Buiten het programmeren met behulp van ObjectWindows heeft Borland nog een andere manier ingebouwd om windows-applicaties te schrijven. Wanneer u oude textbased DOS-applicaties bezit kunt u door middel van het vervangen van 'Uses Crt;' door 'Uses WinCrt;' een windows-applicatie genereren. Let wel, dit kan alleen met textbased programma's. Een nog veel eenvoudigere manier om dan 'Hello World' te schrijven is het volgende:

```
Program Hello;

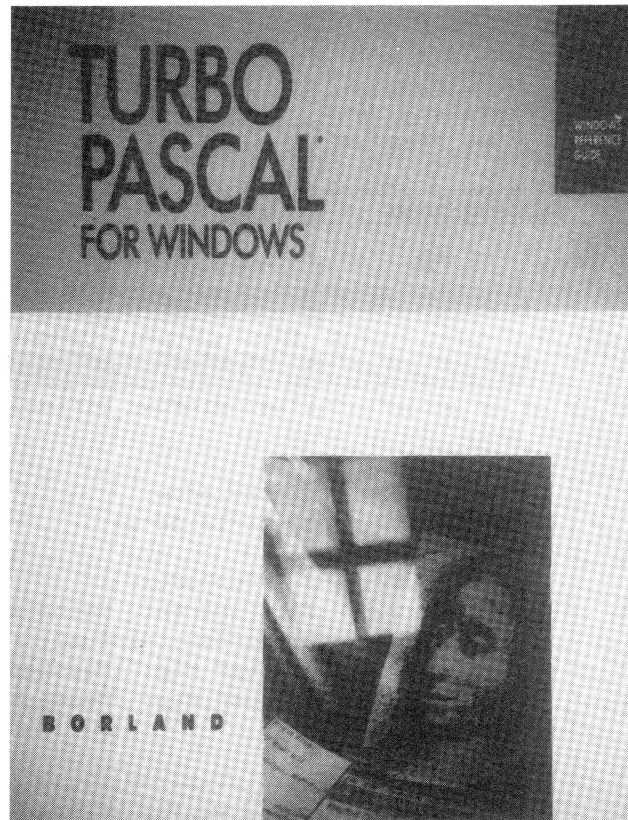
Uses WinCrt;

Begin
 WriteLn ('Hello World');
End.
```

De uitkomst is niet precies hetzelfde als de eerder gegeven 'Hello World', omdat daar niet in het window werd geschreven, terwijl dat hier wel het geval is. Ook hier krijgt u een window dat is te verslepen, te vergroten, te verkleinen, enzovoort. U kunt zelfs door de tekst heenscrollen. Uw oude applicaties zijn dus nog niet verloren zolang ze text-based zijn.

Debugger

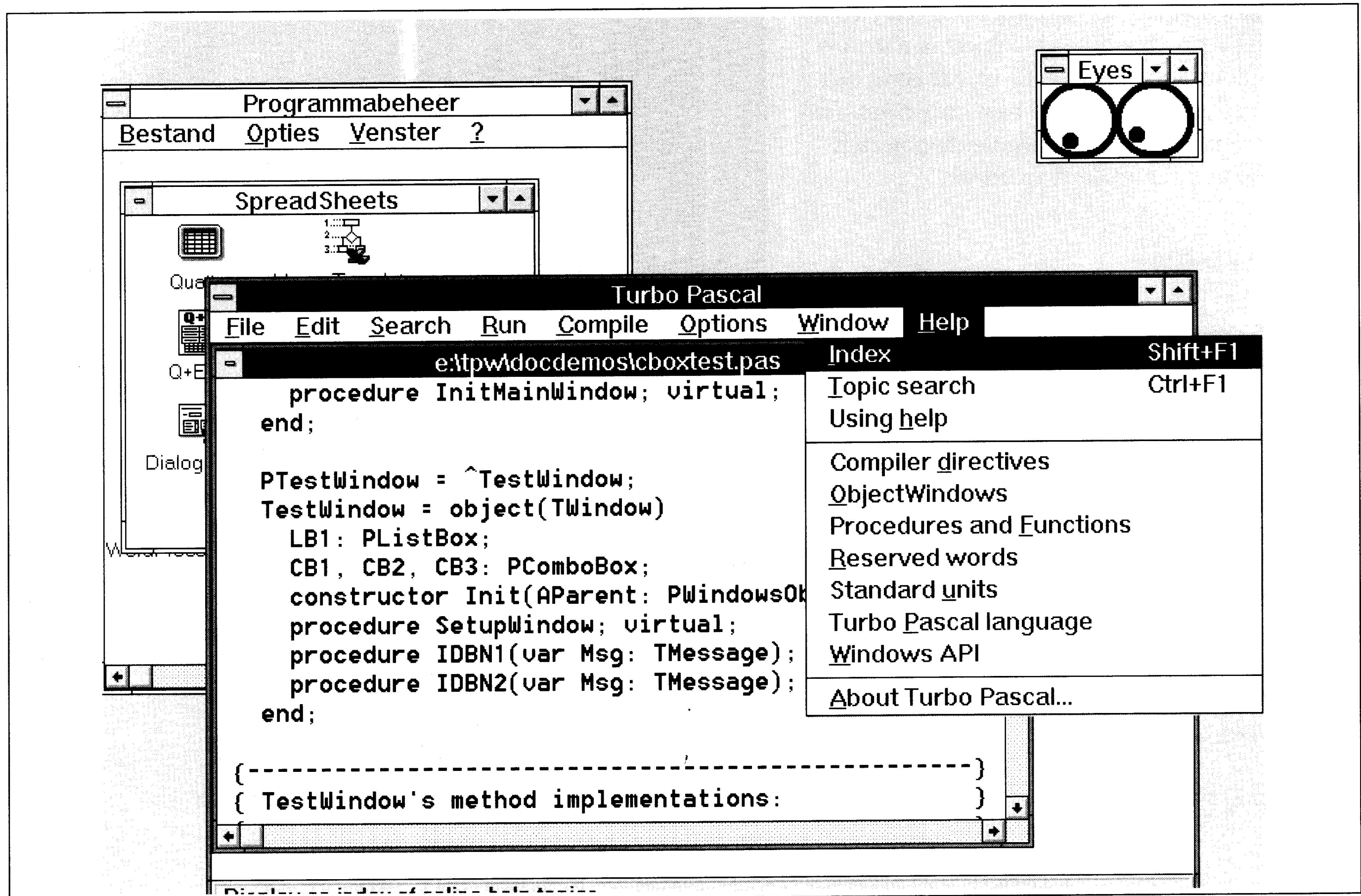
Turbo Pascal voor Windows is in het bezit van een debugger die het debuggen van Windows-applicaties ondersteunt. Dit is het enige aspect van TPW dat niet echt windows-hosted is. Het is namelijk een text-based programma. Omdat Windows low-level applicaties, zoals een debugger en een profiler niet toestaat als echte Windows-applicaties, heeft Borland zijn heil gezocht in het text-based maken van de debugger. Deze is overigens wel alleen op te roepen vanuit Windows. De debugger ondersteunt het gebruik van twee monitors, zodat u op het ene scherm Windows in beeld hebt, terwijl op



het andere scherm de debugger aan het werk is. Het irritante display-swapping is dan verleden tijd. Een andere zeer handige feature die de debugger ondersteunt is backtracing, het teruggaan in het programma. De debugger is zeer uitgebreid en is de speciale Windows-versie van de Turbo Debugger.

WRT

Net als bij Borland C++ wordt ook bij TPW de Whitewater Resource Toolkit meegeleverd. De WRT maakt het mogelijk om resources als bitmaps en dialog boxes te editen zonder dat de source waarin zij voorkomen gehercompileerd hoeft te worden. De WRT kan de benodigde informatie zo uit de .EXE file halen. Andere files die met de WRT bewerkt kunnen worden zonder te hercompileren zijn .RES, .DLL, .H, .ICO, .CUR en .BMP-files. Files die wel opnieuw gecompileerd moeten worden zijn .DLG en .RC-files. Met de WRT is het dus mogelijk om bijvoorbeeld een eigen cursor te tekenen, een about-box te veranderen, een icoon of een dialog box te ontwerpen etcetera. Dit alles zonder dat er ook maar een regel code geschreven hoeft te



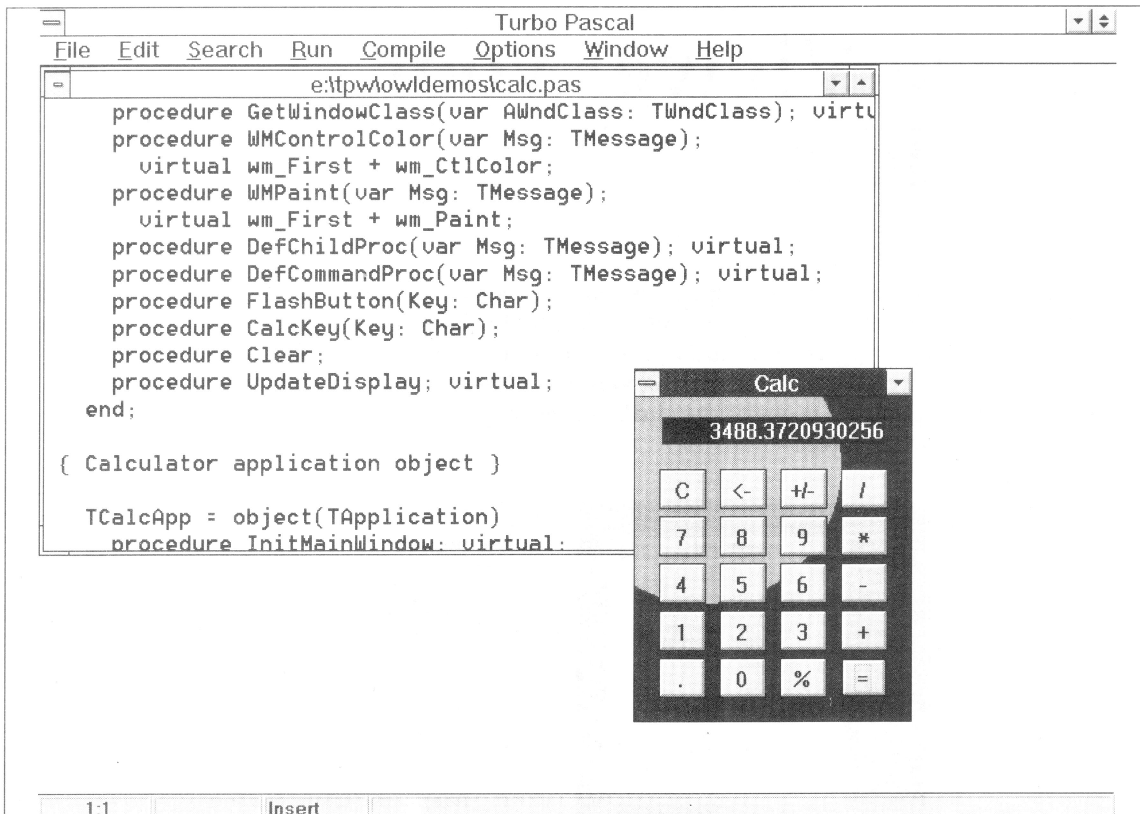
worden. De WRT is niet bij Borland zelf ontwikkeld, maar is gekocht van de Whitewater Group. De WRT is helaas in Actor geschreven, hetgeen impliceert dat het allemaal wat trager gaat. De tool is echter zeer handig.

Windows-mogelijkheden

De applicaties die men met TPW kan schrijven zijn niet beperkt tot 'fancy looking' programma's. TPW ondersteunt bijvoorbeeld het schrijven van Dynamic Link Libraries (DLL's). Dit zijn Windows-specifieke libraries die door meerdere applicaties gebruikt kunnen worden terwijl er maar een zo'n library in het geheugen aanwezig is, de debugger is hier een voorbeeld van. Ook is het mogelijk om applicaties te ontwikkelen die gebruik maken van Dynamic Data Exchange (DDE). Iedere gebruiker van Windows kent wel het clipboard, een plaats waar tijdelijk een hoeveelheid informatie in opgeslagen kan worden die een andere applicatie weer kan opvragen. De listings die bijvoorbeeld in dit artikel staan zijn vanuit TPW via het clipboard naar Word for Windows getransporteerd. Met DDE gebeurt dit achter de schermen. Als zou worden aangegeven dat een bepaalde file uit TPW een dynamische link naar dit artikel zou moeten hebben en een verandering in die file zou worden doorgevoerd, dan zou deze verandering automatisch in het artikel terechtkomen. TPW ondersteunt helaas niet het schrijven van applicaties die gebruik maken van OLE. Dit is een andere manier van dynamische doorvoering van veranderingen die Microsoft in de volgende release van Windows zal verwezenlijken. Met een upgrade van TPW moet dan ook rekening worden gehouden wanneer Microsoft uitkomt met Windows 3.1.

Voordelen

Het schrijven van Windows-applicatie brengt voor de gebruiker en voor de programmeur een aantal voordelen. De gebruiker zal een Windows-applicatie eerder onder



de knie hebben, omdat alle Windows-applicaties dezelfde 'look and feel' hebben. Omdat Windowsapplicaties device-independent zijn hoeft de gebruiker bij het installeren van zo'n applicatie geen lange vragenlijst in te vullen over het gebruikte systeem. Dit voordeel geldt natuurlijk ook voor de programmeur, omdat Windows deze zaken voor haar rekening neemt. Geschreven code werkt dus zonder problemen op een groot aantal computers. Verschillende muizen, printers en monitors worden allemaal door Windows van de programmeur afgeschermd. Windows-specifieke kenmerken als DDE, multitasking en groot geheugengebruik zijn ook voor programmeur en gebruiker van belang.

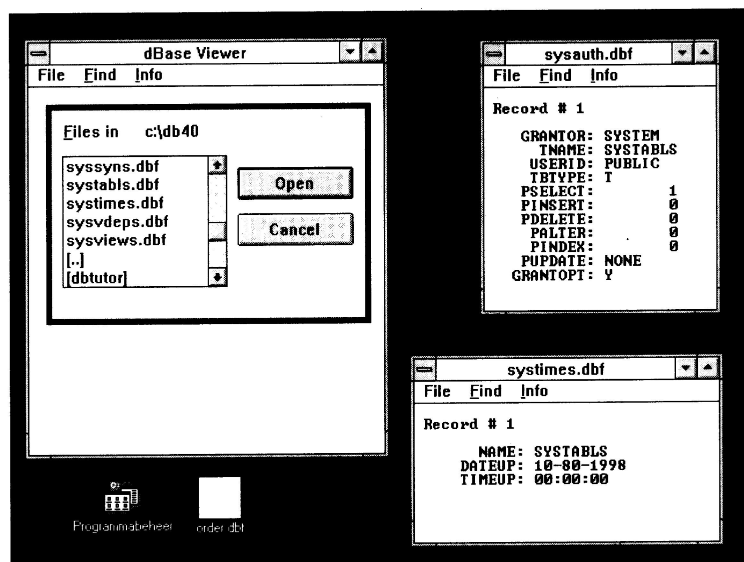
Conclusie

Borland heeft met TPW een duidelijke stap vooruit gezet in het ontwikkelen van Windows-applicaties. TPW maakt het de programmeur een stuk eenvoudiger dan de combinatie van de System Development Kit en een C-compiler. Niet alleen het Windows-hosted zijn van de compiler is een voordeel, ook het meegeleverde framework Object-Windows maakt het een stuk makkelijker om Windows-applicaties te schrijven.

De WRT is natuurlijk een uitermate handige tool en ook de debugger geeft geen reden tot klagen. Door de vernieuwingen die Microsoft zal toepassen in Windows 3.1 wordt het ook mogelijk om een echte Windows-hosted debugger te lanceren, waarna een profiler ook wel te verwachten zal zijn. Wanneer u het prijsverschil ziet van Borland's Turbo Pascal for Windows en de SDK + C-compiler zult u helemaal uw vlucht tot TPW nemen. De laatste combinatie (zonder WRT) kost zo'n tweeduizend gulden, terwijl u TPW voor zo'n vijfhonderd gulden in uw bezit kunt krijgen.

dBase Viewer voor Windows

Hoe mooi de Windows-omgeving ook is om in te werken, men zal toch regelmatig naar de DOS prompt terug moeten om een van die kleine tools te activeren die (nog) niet binnen de Windows-omgeving voor handen zijn. Indien u een intensieve dBase-gebruiker bent kan de in dit artikel opgenomen listing van een dBase viewer u wellicht van dienst zijn. Dit programma geeft de mogelijkheid dBase bestanden te bekijken en hier eenvoudige zoekacties in te verrichten.



Indien u over een 386-computer beschikt is het mogelijk dBase zelf in een Window te draaien; dit pakket is echter dermate geheugen-intensief dat dit op veel systemen problemen geeft. dBview neemt slechts 20 KiloByte in beslag en kan dus zonder problemen naast de grotere applicatie's in Windows actief zijn. Het programma kan bovendien meerdere malen geladen worden zodat het mogelijk wordt records uit verschillende dBase bestanden met elkaar te vergelijken.

De listing van het dBview programma is van een behoorlijke omvang en verdient dus enige toelichting. De opbouw ziet er in pseudocode als volgt uit:

```
WinMain()
  Initialiseer en open het mainwindow
  Zolang het main window geopend is:
    Stuur berichten door naar de
    verschillende window functie's
```

```
WndProc( bericht )
  Doe bericht:
    PAINT:
      Laat de inhoud van het huidige
      record zien in het main window
    KEYDOWN:
      Huidige record wordt
        Home - Eerste record
        PgUp - Vorige record%
        PgDn - Volgende record
        End  - Laatse record
      Stuur PAINT bericht
    OPEN:
      Start de Open dialog
      Laad het dBase bestand
      Maak eerste record de huidige
      Stuur PAINT bericht
    FIND:
      Start de Find dialog
      Stuur PAINT bericht
    INFO:
      Start de Info dialog
    QUIT:
      Programma afsluiten
```

Net als de main() functie in standaard C programma's moet iedere Windows applicatie een WinMain() functie bevatten. Deze functie controleerd eerst op eventuele eerdere instanties van het programma.

```
if ( !hPrevInstance )
{
  // Initialiseer applicatie
  WndClass.hCursor = LoadCursor(
  ...
  registerClass(&WndClass);
}
```

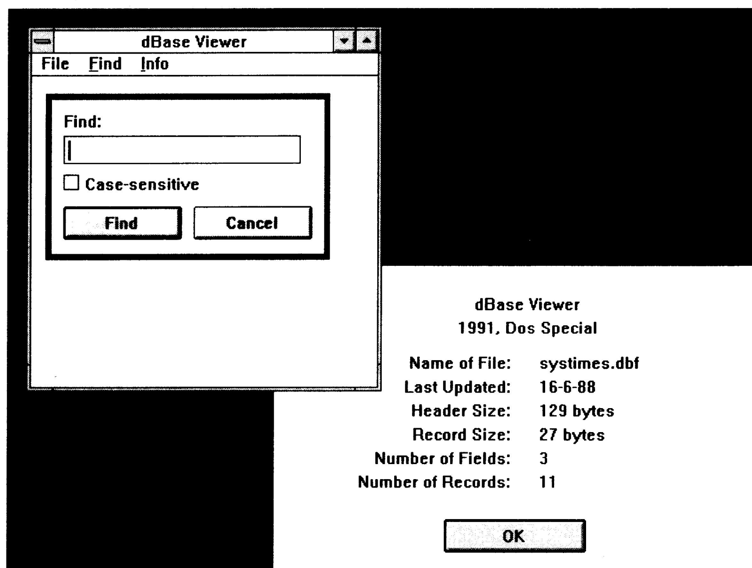
Aangezien alle instanties van het programma van dezelfde window class gebruik kunnen maken, hoeft deze class slechts bij de eerste instantie geregistreerd te worden. Vervolgens wordt een main window gecreëerd met de titel: "dBase Viewer" en het programma komt in een message-loop terecht.
Alle berichten bedoeld voor de dBview applicatie worden in deze loop door DispatchMes-

sage() verspreid over de verschillende window functie's.

```
while ( GetMessage(&msg, NULL, 0, 0) )
{
  TranslateMessage( &msg );
  DispatchMessage( &msg );
}
```

Zo zullen de berichten voor het main window doorgestuurd worden naar WndProc() en de berichten voor de dialog boxes naar de betreffende dialog functies.

Het programma is geschreven in de Borland C++ omgeving, maar is met weinig aanpassing geschikt te maken voor bijvoorbeeld de Microsoft 6.0 / SDK combinatie.




```
;DBVIEW.DEF

NAME            DBVIEW
DESCRIPTION     'dbview'
EXETYPE         WINDOWS
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        8192
STACKSIZE       5120
EXPORTS         WndProc        @1
       OpenDlg        @2
       FindDlg        @3
       InfoDlg        @4

;EOF DBVIEW.DEF
```

```
// DBVIEW.H

#define IDM_QUIT  101
#define IDM_ABOUT 102
#define IDM_OPEN  103
#define IDM_FIND  201
#define IDM_INFO  301

//---------------------------------------------------


#define IDC_FILES   401
#define IDC_PATH    402
#define IDC_LISTBOX 403

#define IDC_EDIT 501
#define IDC_CASE 502

#define IDC_FILENAME 601
#define IDC_LASTUPD  602
#define IDC_HSIZE    603
#define IDC_RSIZE    604
#define IDC_FIELDS   605
#define IDC_RECORDS  606

//---------------------------------------------------


#define szAppName "dbviewClass"

#define FIND    1
#define INFO    2

// Maximum aantal velden per record.
```

```
#define MAXFIELDS 20
#define MAXFIND   20

// Header info van het dBase bestand.
struct DBHEADER {
  unsigned char Version;
  char Year, Month, Day;
  long RecCount;
  int  HeaderSize;
  int  RecSize;
  char Reserved1[3];
  char LanInfo[13];
  char reserved2[4];
};

// Field info van het dBase bestand.
struct DBFIELD {
  char Name[11], Type;
  long Address;
  unsigned char Length;
  char Decimals;
  char LanInfo[2];
  char WorkArea;
  char Reserved1[2];
  char SetFieldsFlag;
  char Reserved2[8];
};

//---------------------------------------------------


long FAR PASCAL WndProc(HWND,unsigned,WORD,LONG);
void dbPaint(HWND);
void dbLoad(HWND);
HANDLE FAR PASCAL OpenDlg(HWND,unsigned,WORD,
LONG);
void UpdateListBox(HWND);
void SeparateFile(LPSTR,LPSTR,LPSTR);
BOOL FAR PASCAL FindDlg(HWND,WORD,WORD,LONG);
BOOL FindStr(PSTR);
BOOL FAR PASCAL InfoDlg(HWND,WORD,WORD,LONG);
void DToString(PSTR,PSTR);
void MToString(PSTR,PSTR);
void LToString(PSTR,PSTR);
void CToString(PSTR,PSTR,unsigned char);
void Error(HWND,LPSTR);

// EOF DBVIEW.H
```

```
// DBVIEW.RC

#include <windows.h>
#include "dbview.h"

//-----------------------------------------------


dbMenu MENU
BEGIN
  POPUP "File"
  BEGIN
    MENUITEM "&Open", IDM_OPEN
    MENUITEM "E&xit", IDM_QUIT
  END
  MENUITEM "&Find",   IDM_FIND
  MENUITEM "&Info",   IDM_INFO
END

//-----------------------------------------------


Open DIALOG 10, 10, 135, 90
STYLE WS_POPUP | WS_DLGFRAME
BEGIN
  LTEXT "&Files in",    IDC_FILES,    5, 5, 32,10
  LTEXT "",             IDC_PATH,    40, 5,100,10
  LISTBOX,              IDC_LISTBOX,  5,20,
70,56,
WS_TABSTOP|WS_VSCROLL
  DEFPUSHBUTTON "Open", IDOK,        80,25, 50,14
  PUSHBUTTON "Cancel",  IDCANCEL,    80,45, 50,14
END

//-----------------------------------------------


FIND DIALOG 10,10,115,65
STYLE WS_POPUP | WS_DLGFRAME
BEGIN
  LTEXT "Find:"                    -1, 5, 5, 40, 8
  EDITTEXT                 IDC_EDIT,
5,15,100,12,
  ES_AUTOHSCROLL
  CHECKBOX "Case-sensitive",IDC_CASE, 5,30,
80,10,
  BS_AUTOCHECKBOX
  DEFPUSHBUTTON "Find",          IDOK, 5,45, 50,14
  PUSHBUTTON "Cancel",     IDCANCEL,60,45,
50,14,
END

//-----------------------------------------------


INFO DIALOG 10,10,200,125
STYLE WS_POPUP | WS_DLGFRAME
BEGIN
  CTEXT "dBase Viewer"          -1, 10, 10,190, 8
  CTEXT "1991, Dos Special"     -1, 10, 20,190, 8
  RTEXT "Name of File: "        -1, 10, 35, 90, 8
  RTEXT "Last Updated: "        -1, 10, 45, 90, 8
  RTEXT "Header Size: "         -1, 10, 55, 90, 8
  RTEXT "Record Size: "         -1, 10, 65, 90, 8
  RTEXT "Number of Fields: "    -1, 10, 75, 90, 8
  RTEXT "Number of Records: "   -1, 10, 85, 90, 8
  LTEXT "",           IDC_FILENAME,110, 35, 90, 8
  LTEXT "",            IDC_LASTUPD,110, 45, 90, 8
  LTEXT "",              IDC_HSIZE,110, 55, 90, 8
  LTEXT "",              IDC_RSIZE,110, 65, 90, 8
  LTEXT "",             IDC_FIELDS,110, 75, 90, 8
  LTEXT "",            IDC_RECORDS,110, 85, 90, 8
  DEFPUSHBUTTON "OK"           IDOK, 70,105,
60,14,
END

// EOF DBVIEW.RC
```

```
// DBVIEW.C

#include <windows.h>
#include <stdio.h>
#include <string.h>
#include "dbview.h"

//-----------------------------------------------


char str[255];
char buf[255];

// Handle naar huidige dBase bestand.
FILE *dbHandle;

// Handle naar de buffer waar het huidige
// record in opgeslagen wordt.
HANDLE hRecBuf = NULL;

// Positie van het eerste record in het
// dBase-bestand.
HANDLE FirstRecPos = NULL;

struct DBHEADER dbHeader;
struct DBFIELD dbField[MAXFIELDS];

int nrFields;
long CurRecord = 0;
int dbLoaded = 0;

//-----------------------------------------------

static HANDLE hInst;

#pragma argsused
int PASCAL WinMain(HANDLE hInstance,
                   HANDLE hPrevInstance,
                   LPSTR lpszCmdLine,
                   int nCmdShow)
{
  WNDCLASS   WndClass;
  HWND       hWnd;
  MSG        msg;

  hInst = hInstance;

  // Registreer alleen de eerste instantie
  // van het programma.

  if (!hPrevInstance)
  {
    // Definieer de nieuwe Window class.

    WndClass.hCursor       = LoadCursor(NULL,
       IDC_ARROW);
    WndClass.hIcon         = NULL;
    WndClass.cbClsExtra    = 0;
    WndClass.cbWndExtra    = 0;
    WndClass.lpszMenuName  = "dbMenu";
    WndClass.lpszClassName= szAppName;
    WndClass.hbrBackground= GetStockObject(
      WHITE_BRUSH);
    WndClass.hInstance     = hInstance;
    WndClass.style         =
CS_VREDRAW|CS_HREDRAW;
    WndClass.lpfnWndProc   = WndProc;

    // Registreer de nieuwe class in Windows

    if (!RegisterClass(&WndClass))
      return FALSE;
  }

  hWnd = CreateWindow(
    szAppName,              // Eigenaar
    "dBase Viewer",         // Naam
    WS_OVERLAPPEDWINDOW,    // Type
    CW_USEDEFAULT,          // X-positie
    CW_USEDEFAULT,          // Y-positie
```

```
    CW_USEDEFAULT,          // Breedte
    CW_USEDEFAULT,          // Hoogte
    NULL,                   // ParentWindow
    NULL,                   // Gebruik
lpszMenuName
    hInstance,              // hInstance
    NULL);                  // lpParam

  ShowWindow( hWnd, nCmdShow );
  UpdateWindow( hWnd );

  // De Main message loop.

  while ( GetMessage( &msg, NULL, 0, 0) )
  {
    TranslateMessage( &msg );
    DispatchMessage( &msg );
  }

  free(hRecBuf);
  fclose(dbHandle);

  // Terug naar Windows.

  return (int) msg.wParam;
}

//---------------------------------------------------

HFONT  hFont;

long FAR PASCAL WndProc (HWND hWnd,
                         unsigned message,
                         WORD wParam, LONG
lParam)
{
  FARPROC lpOpenDlg, lpFindDlg, lpInfoDlg;
  BOOL Found;

  switch(message)
  {
    case WM_CREATE:
      // Gebruik een Fixed font voor TextOut().
      hFont = GetStockObject(OEM_FIXED_FONT);
      break;

    case WM_PAINT:
      if (!IsIconic(hWnd))
dbPaint(hWnd);
      else
return DefWindowProc(hWnd,message,
     wParam,lParam);
      break;

    case WM_KEYDOWN:
      switch ( wParam )
      {
case VK_HOME:  // Home
        CurRecord = 0;
        break;

case VK_END:   // End
        CurRecord = dbHeader.RecCount-1;
        break;

case VK_PRIOR: // PgUp
  if (CurRecord > 0) CurRecord--;
        break;

case VK_NEXT:  // PgDn
  if (CurRecord < dbHeader.RecCount-1)
          CurRecord++;
        break;

default:
  return DefWindowProc(hWnd,message,
     wParam,lParam);
      }

      // Genereer een WM_PAINT message.
      InvalidateRect(hWnd, NULL, TRUE);
      UpdateWindow(hWnd);
      break;

    case WM_COMMAND:
      // Een dbMenu optie is geactiveerd.
      switch( wParam )
      {
case IDM_OPEN:
  fclose(dbHandle);

  lpOpenDlg = MakeProcInstance(
(FARPROC) OpenDlg, hInst);
        dbHandle = (FILE *) DialogBox(hInst,
                   "Open", hWnd, lpOpenDlg);
  FreeProcInstance(lpOpenDlg);

  dbLoaded = 0;
  if (dbHandle) dbLoad(hWnd);

  CurRecord = 0;
  // Genereer een WM_PAINT message.
  InvalidateRect(hWnd, NULL, TRUE);
        UpdateWindow(hWnd);
        break;

case IDM_QUIT:
        PostMessage(hWnd, WM_CLOSE, 0, 0L);
        break;

      case IDM_FIND:
  lpFindDlg = MakeProcInstance(FindDlg,
     hInst);
  // Start een Find-dialog.
        Found = DialogBox(hInst,
    MAKEINTRESOURCE(FIND),
    hWnd, lpFindDlg);
  FreeProcInstance(lpFindDlg);

  if (Found) {
    // Genereer een WM_PAINT message.
    InvalidateRect(hWnd, NULL, TRUE);
    UpdateWindow(hWnd);
  }
        break;

      case IDM_INFO:
        lpInfoDlg = MakeProcInstance(
InfoDlg, hInst);
  // Start een Info-dialog.
        DialogBox(hInst,MAKEINTRESOURCE(INFO),
                  hWnd, lpInfoDlg);
  FreeProcInstance(lpInfoDlg);
        break;
      }
      break;

    case WM_DESTROY:
      PostQuitMessage( 0 );
      break;

    // Overige messages worden afgehandeld door
    // Windows zelf.
    default:
      return DefWindowProc(hWnd,message,
                           wParam,lParam);
  }
  return (long) 0;
}

//---------------------------------------------------


void dbPaint(HWND hWnd)
{
  TEXTMETRIC tm;
  COLORREF OldColor, Grey;
  PAINTSTRUCT ps;
  short int FldPosX, FldPosY;
  int CharH, CharW, i, j;
```

```
  PSTR CurField,strPtr;
  int MaxLength;
  RECT wRect;

  HDC hDC = BeginPaint( hWnd, &ps );

  // Selecteer een fixed font en bewaar de
  // vorige font.
  HFONT hOldFont = SelectObject(hDC,hFont);

  // Bepaal de character afmetingen van
  // de nieuwe font.
  GetTextMetrics(hDC,&tm);
  CharH = tm.tmHeight;
  CharW = tm.tmAveCharWidth;

  // Bepaal de maximum text lengte afhankelijk
  // van de huidige window afmetingen.
  GetClientRect( hWnd, &wRect );
  MaxLength = (wRect.right-(14*CharW)) / CharW;

  FldPosX = 13*CharW;
  FldPosY = CharH;

  Grey = RGB(200,200,200);

  if ( dbLoaded )
  {
    if ( dbHeader.RecCount )
    {
      fseek(dbHandle, FirstRecPos +
    dbHeader.RecSize * CurRecord,
    SEEK_SET);
      fread((void *) hRecBuf,
    dbHeader.RecSize, 1, dbHandle);
    }

    sprintf(str, "Record # %d", CurRecord+1);
    TextOut(hDC, CharW, FldPosY, str,
    strlen(str));

    FldPosY = 3*CharH;
    CurField = (char *) hRecBuf;

    for (i=0; i<nrFields && i<MAXFIELDS; i++)
    {
      // Veld naam:
      strncpy(str, dbField[i].Name, 10);
      str[10] = 0;
      strcat(str,":");
      SetTextAlign(hDC,TA_RIGHT);
      OldColor = SetTextColor(hDC,Grey);
      TextOut(hDC, 12*CharW, FldPosY, str,
      strlen(str));

      // Veld inhoud:
      if ( dbHeader.RecCount )
      {
SetTextAlign(hDC,TA_LEFT);
SetTextColor(hDC,OldColor);

switch (dbField[i].Type)
{
  case 'D':   // Date
    DToString(str,CurField);
    break;
  case 'M':   // Memo
    MToString(str,CurField);
    break;
  case 'L':   // Logical
    LToString(str,CurField);
    break;
  case 'N':   // Numeric
  case 'C':   // Character
    CToString(str,CurField,
      dbField[i].Length);
    break;
}

// Formateer veld inhoud indien langer
// dan MaxLength.
if (MaxLength > 0 && str[0])
{
  strPtr = str;
  for (j=1; j<5; j++)
  {
    strncpy(buf,strPtr,MaxLength);
    buf[MaxLength] = 0;
    TextOut(hDC,FldPosX,FldPosY,buf,
    strlen(buf));
    FldPosY += CharH;
    if (strlen(strPtr) < MaxLength)
      break;
    strPtr = &str[MaxLength*j];
  }
}
else FldPosY += CharH;
CurField += dbField[i].Length;
      }
      else FldPosY += CharH;
    }
  }
  // !dbLoaded
  else SetWindowText(hWnd, "dBase Viewer");

  SelectObject(hDC,hOldFont);
  EndPaint( hWnd, &ps );
}

//--------------------------------------------------

char OpenName[128];
char DefPath[128];
char DefSpec[13] = "*.dbf";
char DefExt[] = ".dbf";

void dbLoad(HWND hWnd)
{
  int i;

  SetWindowText(hWnd, OpenName);

  // Lees de Header informatie van de database.
  fread((void *) &dbHeader,
 sizeof(struct DBHEADER), 1, dbHandle);

  // Bepaal aantal velden per record.
  nrFields = (dbHeader.HeaderSize-
      sizeof(struct DBHEADER) -1) /
      sizeof(struct DBFIELD);

  // Lees de veld informatie in.
  for ( i=0; i<nrFields && i<MAXFIELDS; i++ )
    fread((void *) (dbField+i),
  sizeof(struct DBFIELD), 1, dbHandle);

  FirstRecPos = sizeof(struct DBHEADER) +
sizeof(struct DBFIELD) *
nrFields + 1;

  // Dealloceer eventueel eerst een vorige
  // record buffer.
  if (hRecBuf != NULL) free(hRecBuf);

  // Alloceer geheugen voor een record buffer.
  hRecBuf = malloc(dbHeader.RecSize);
  if ( hRecBuf==NULL )
  {
    Error(hWnd,"Memory allocation failure");
    dbLoaded = 0;
  }
  else dbLoaded = 1;
}

//--------------------------------------------------


HANDLE FAR PASCAL OpenDlg(HWND hDlg,
                          unsigned message,
  WORD wParam,LONG lParam)
```

```
{
  HANDLE hFile=NULL;

  switch (message)
  {
    case WM_COMMAND:
    switch (wParam)
    {
      case IDC_LISTBOX:
        switch (HIWORD(lParam))
        {
  case LBN_SELCHANGE:
    // Indien selectie een directory naam:
    // voeg "*.dbf" toe.
    if (DlgDirSelect(hDlg, OpenName,
                IDC_LISTBOX))
      strcat(OpenName, DefSpec);
            break;

  // List item geselecteerd
  case LBN_DBLCLK:
            goto openfile;
        }
        return TRUE;

      case IDOK:  // OK Button geactiveerd.

openfile:

if ( strchr(OpenName, '*') )
{
  SeparateFile((LPSTR) str,(LPSTR)DefSpec,
       (LPSTR) OpenName);
  if (str[0]) strcpy(DefPath,str);
  UpdateListBox(hDlg);
  return TRUE;
}
hFile = (HANDLE) fopen(OpenName, "r+b");
if (!hFile)
  Error(hDlg,"Open failure");

// Open Dialog afsluiten
EndDialog(hDlg, hFile);
return TRUE;

      case IDCANCEL:
// Cancel Open Dialog
EndDialog(hDlg, NULL);
return FALSE;
    }
    break;

    case WM_INITDIALOG:
UpdateListBox(hDlg);
return FALSE;
  }
  return FALSE;
}

//-------------------------------------------------

// List directories en files met de
// DefSpec (=*.dbf) extensie.

void UpdateListBox(HWND hDlg)
{
  strcpy(str, DefPath);
  strcat(str, DefSpec);
  DlgDirList(hDlg, str, IDC_LISTBOX,
      IDC_PATH, 0x4010);

  if (!strchr (DefPath, ':'))
      DlgDirList(hDlg, DefSpec, IDC_LISTBOX,
 IDC_PATH, 0x4010);

  if (strstr (DefPath, ".."))
      DefPath[0] = 0;
}

//-------------------------------------------------
```

```
// Splits path en filenaam

void SeparateFile(LPSTR lpDestPath,
  LPSTR lpDestFileName,
  LPSTR lpSrcFileName)
{
  LPSTR lpTmp;
  char  cTmp;

  // Begin bij laatste character
  lpTmp = lpSrcFileName +
  (long) lstrlen(lpSrcFileName);

  while (*lpTmp != ':' && *lpTmp != '\\'
       && lpTmp > lpSrcFileName)
    // Voorafgaande character
    lpTmp = AnsiPrev(lpSrcFileName, lpTmp);

  if (*lpTmp != ':' && *lpTmp != '\\')
  {
    lstrcpy(lpDestFileName, lpSrcFileName);
    lpDestPath[0] = 0;
    return;
  }

  lstrcpy(lpDestFileName, lpTmp + 1);
  cTmp = *(lpTmp + 1);
  lstrcpy(lpDestPath, lpSrcFileName);
  *(lpTmp + 1) = cTmp;
  lpDestPath[(lpTmp - lpSrcFileName) + 1] = 0;
}

//-------------------------------------------------

BOOL CaseSensitive=0;

#pragma argsused
BOOL FAR PASCAL FindDlg(HWND hDlg,WORD message,
WORD wParam,LONG lParam)
{
  if ( message==WM_INITDIALOG)
  {
    // Case-box is disabled.
    CaseSensitive = 0;
    return TRUE;
  }

  if ( message==WM_COMMAND)
    switch (wParam)
    {
      case IDOK:
        if (dbLoaded)
        {
  GetDlgItemText(hDlg, IDC_EDIT,
 str, MAXFIND);
  if ( FindStr(str) )
  {
    // String gevonden!
    // Find Dialog afsluiten
    EndDialog(hDlg,TRUE);
            return TRUE;
          }
}
Error(hDlg,"Not found");

// String niet gevonden!
// Find Dialog afsluiten
EndDialog(hDlg, FALSE);
return TRUE;

      case IDCANCEL:
// Cancel Find Dialog.
EndDialog(hDlg,FALSE);
return TRUE;

      case IDC_CASE:
// Case CheckBox clicked.
CaseSensitive = !CaseSensitive;
return TRUE;
```

```
    }
  return FALSE;
}
//----------------------------------------------

BOOL FindStr(PSTR string)
{
  int RecPos,Found;

  // Zoek vanaf het volgende record.
  long rec = CurRecord+1;

  while (rec < dbHeader.RecCount)
  {
    fseek(dbHandle, FirstRecPos +
  dbHeader.RecSize * rec, SEEK_SET);
    fread((void *) hRecBuf,
  dbHeader.RecSize, 1, dbHandle);

    // Start zoek actie in eerste veld
    RecPos = 0;
    while ( RecPos < dbHeader.RecSize )
    {
      if ( CaseSensitive )
Found = !strncmp(string, (PSTR)
 (hRecBuf+RecPos),
 strlen(string));
      else // !CaseSensitive
Found = !strnicmp(string, (PSTR)
  (hRecBuf+RecPos),
  strlen(string));
      if ( Found )
      {
// string gevonden!
CurRecord = rec;
return TRUE;
      }
      RecPos++;
    }
    // Zoek verder in volgende record.
    rec++;
  }

  // String niet gevonden!
  return FALSE;
}
//----------------------------------------------

#pragma argsused
BOOL FAR PASCAL InfoDlg(HWND hDlg,WORD message,
WORD wParam,LONG lParam)
{
  if ( message==WM_INITDIALOG )
  {
    if ( dbLoaded )
    {
      SetDlgItemText(hDlg,IDC_FILENAME,OpenName);

      sprintf(str,"%d-%d-%d",dbHeader.Day,
      dbHeader.Month, dbHeader.Year);
      SetDlgItemText(hDlg, IDC_LASTUPD, str);

      itoa(dbHeader.HeaderSize, str, 10);
      strcat(str, " bytes");
      SetDlgItemText(hDlg, IDC_HSIZE, str);

      itoa(dbHeader.RecSize, str, 10);
      strcat(str, " bytes");
      SetDlgItemText(hDlg, IDC_RSIZE, str);

      itoa(nrFields, str, 10);
      SetDlgItemText(hDlg, IDC_FIELDS, str);

      ltoa(dbHeader.RecCount, str, 10);
      SetDlgItemText(hDlg, IDC_RECORDS, str);
    }
      return TRUE;
  }

  else if (message==WM_COMMAND && wParam==IDOK)
  {
    // Info Dialog afsluiten.
    EndDialog(hDlg, TRUE);
    return TRUE;
  }
  return FALSE;
}
//----------------------------------------------

// Date naar string conversie

void DToString(PSTR String,PSTR Date)
{
  String[0] = Date[6];
  String[1] = Date[7];
  String[2] = '-';
  String[3] = Date[4];
  String[4] = Date[5];
  String[5] = '-';
  String[6] = '1';
  String[7] = '9';
  String[8] = Date[2];
  String[9] = Date[3];
  String[10] = 0;
}
//----------------------------------------------

#pragma argsused
void MToString(PSTR String,PSTR Memo)
{
  strcpy(String,"** Memo **");
}
//----------------------------------------------

// Logical naar string conversie

void LToString(PSTR String,PSTR Fld)
{
  if ( Fld[0] ) String[0] = 'T';
  else String[0] = 'F';
  String[1] = 0;
}
//----------------------------------------------

// Character naar string conversie

void CToString(PSTR String, PSTR Fld,
       unsigned char Length)
{
  int i=0;
  while (i < Length)
  {
    String[i] =Fld[i+1];
    i++;
  }
  str[i] = 0;
}
//----------------------------------------------

void Error(HWND hWnd, LPSTR lptext)
{
  MessageBox(hWnd,lptext,"dbview",MB_OK |
  MB_ICONHAND);
}

// EOF DBVIEW.C
```

Eiffel

Eiffel is een object-georiënteerde programmeertaal, ontwikkeld rondom begrippen als software hergebruik, uitbreidbaarheid en compatibiliteit. De ontwikkelaar van Eiffel, Bertrand Meyer, heeft bovendien veel aandacht besteed aan overdraagbaarheid, correctheid en robuustheid van de taal. Al deze eigenschappen komen min of meer voort uit de object-georiënteerde achtergrond van de Eiffel-taal. We zullen ons in dit artikel richten op de syntactische aspecten van de Eiffel programmeertaal, alsmede de sterke punten van de taal.

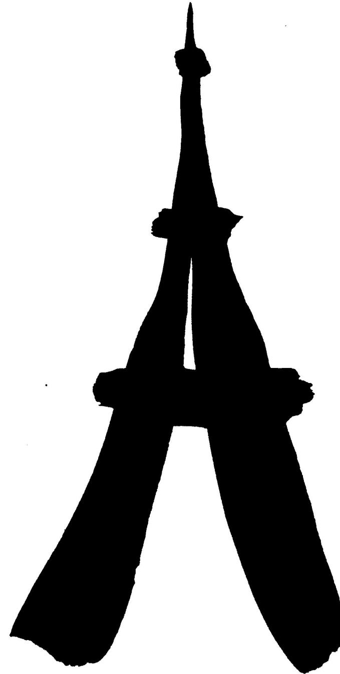
Net als bij sommige C++ implementaties wordt ook Eiffel-code omgezet in C-code, die vervolgens met een reguliere C-compiler te verwerken is tot een uitvoerbaar programma. Dit heeft als voordeel dat Eiffel-programma's op een breed scala aan machines gebruikt kunnen worden, omdat voor de meeste platforms wel een C-compiler beschikbaar is. Een groot verschil met C++ is echter dat Eiffel een compleet andere syntax heeft dan C, en dus niet direct compatible is met deze taal.

De Eiffel-taal is een van de meest complete object-georiënteerde talen van het moment waarin bruikbare applicaties kunnen worden geschreven. In tegenstelling tot de huidige versie van C++ worden genericiteit en exception handling ondersteund, terwijl ook pre- en postcondities kunnen worden gesteld voor procedures. De syntax van Eiffel lijkt een beetje op die van Ada en Pascal.

De leden van een class heten in Eiffel-syntax features; dit zijn zowel data-members als member-functies en -procedures. Met behulp van het export-keyword wordt aangegeven welke leden de publieke interface (naar de buitenwereld) verzorgen.

Een eenvoudige class om een twee-dimensionaal punt weer te geven, zou in Eiffel als volgt worden gedefinieerd:

```
class POINT
  export
    draw
  feature
    x, y : INTEGER;
      — Functie om punt te tekenen
    draw is
      do
        — Code om punt te tekenen
      end; — draw
end — POINT
```



Twee liggende streepjes (—) worden gebruikt om commentaar in te luiden tot het einde van de regel. De members x en y geven de coördinaten van het punt aan, terwijl draw de routine is die het punt zal tekenen. Zoals uit het export-gedeelte blijkt, wordt alleen draw bekend gemaakt aan de buitenwereld; de x en y members blijven verborgen in de class. Een POINT instantie wordt als volgt aangemaakt:

```
my_point : POINT;
```

Nu is een variabele (eigenlijk een referentie) van het type POINT gecreëerd. Met de voorgedefinieerde routine Create wordt een punt aangemaakt met de waarde (0,0):

```
my_point.Create;
```

Natuurlijk moeten we een Create-routine hebben die twee integers als argumenten heeft, teneinde een punt ook te kunnen initialiseren. Hiertoe wordt binnen de class (dus analoog aan de definitie van de draw-routine) een Create-routine gedefinieerd:

```
class POINT
  -- ... als boven ...

  Create(a, b : INTEGER) is
    do
      x := a;
      y := b
    end;  -- Create

end -- POINT
```

Nu kan de variabele my_point als volgt geïnitialiseerd worden met de waarde (50,415):

```
my_point.Create(50,415);
```

Het voorgedefinieerde woord Result kan gebruikt worden om een functie een waarde te laten terugkeren. Zo kunnen we bijvoorbeeld de functie at_origin (True als een punt op de oorsprong ligt) van het type BOOLEAN als volgt schrijven voor de POINT class:

```
class POINT
  export
    draw, at_origin
  -- ... als boven ...

  at_origin: BOOLEAN is
    do
      Result := (x = 0) and (y = 0)
    end;  -- at_origin

end  -- POINT


p : POINT;

  ...
  if p.at_origin then
    ...
```

Met de definitie p : POINT wordt nog geen object gecreëerd, maar een referentie (op dat moment nog void) naar een later te definiëren object POINT. Dit houdt in dat een object expliciet gedefinieerd moet worden met behulp van de Create-routine. Pas dan wordt p aan een bestaand object gekoppeld, waarna ermee gewerkt kan worden. Indien een referentie losgekoppeld moet worden van een object, kan de routine Forget gebruikt worden:

```
p.Forget;
```

Een referentie kan dus in twee toestanden verkeren: in VOID-toestand (nadat de referentie gedeclareerd is) en in CREATED-toestand (nadat de Create-routine aangeroepen is). Met Forget keert de referentie weer terug in de VOID-toestand.

Met de routine Clone kunnen referenties ook in CREATED-toestand gebracht worden:

```
p, q : POINT;

q.Create(20,89);  -- creatie van q

p.Clone(q);  -- creatie van p aan de hand
van q
```

Alle velden van het POINT-argument (in het voorbeeld dus q.x en q.y) worden hierbij naar de corresponderende velden van het object van waaruit Clone werd aangeroepen (dus de variabele p) gekopieerd. Hetzelfde effect wordt trouwens bereikt met p:=q.

Inheritance

Natuurlijk ondersteunt Eiffel single inheritance, alsmede multiple en repeated inheritance. Inheritance (of overerving) betekent dat een nieuwe class-definitie wordt aangemaakt die de eigenschappen erft van een reeds bestaande class. Op deze manier kan de code van de ouder-class (ook wel basis- of superclass genoemd) opnieuw gebruikt worden.

We zullen een class CIRCLE definiëren die gebaseerd is op een POINT. Een cirkel kan immers gezien worden als een punt met als extra eigenschap een variabele die de straal van de cirkel aangeeft; de (x,y) members van de ouder-class POINT worden dan gebruikt om het middelpunt van de cirkel aan te geven. De definitie van CIRCLE zou er in Eiffel als volgt uitzien:

```
class CIRCLE
  export
    draw, at_origin
  inherit
    POINT
  redefine
    draw
  feature
    radius : INTEGER;
    draw is
      do
        -- Code om cirkel te tekenen
      end;
end  -- CIRCLE
```

Het inherit POINT geeft aan dat POINT een basis-class is voor CIRCLE; redefine draw betekent dat de draw-functie in de CIRCLE-class opnieuw gedefinieerd wordt. In tegenstelling tot C++ is dit noodzakelijk in Eiffel. De at_origin-functie van POINT wordt ongewijzigd geërfd in de CIRCLE-class. Overigens zou de CIRCLE-class ook een Create-functie moeten hebben.

Polymorfie

Het compatibiliteitsprincipe van basis- en afgeleide classes betekent in Eiffel dat een referentie aan een basis-class altijd kan refereren aan een afgeleide class; immers alle eigenschappen die dan gebruikt worden (de members van de basis-class) zijn ook beschikbaar voor de afgeleide class. In het volgende voorbeeld laten we een POINT-referentie p wijzen naar een CIRCLE:

```
p : POINT;
c : CIRCLE;

c.Create(...);
p := c;  — Laat p refereren aan c

p.draw;  — Teken p
```

Dankzij de dynamische binding die Eiffel zal toepassen, zal de draw-routine van CIRCLE worden aangeroepen, hoewel p statisch als POINT-referentie is gedeclareerd. Dit wordt ook wel polymorfie (veelvormigheid) genoemd en is een belangrijke eigenschap van object-georiënteerde systemen.

Abstracte basis-classes

In sommige gevallen is het wenselijk dat een basis-class alleen dient om er classes van af te leiden, en zelf geen concrete betekenis heeft. Dit is bijvoorbeeld het geval bij een SHAPE-class, die als basis-class zou kunnen fungeren voor POINT en CIRCLE. Van het type SHAPE wordt nooit een instantie gemaakt met behulp van Create of Clone. Een routine als draw moet dus in elke afgeleide class opnieuw gedefinieerd worden, omdat draw voor SHAPE geen betekenis heeft. Om dit aan te geven kan de draw-routine in Eiffel deferred (uitgesteld) worden gedeclareerd:

```
class SHAPE
  export
    — ... Publieke interface
  feature
    — ... Definitie van de members

    draw is
      deferred
      end;

end  — SHAPE
```

Dit betekent dat de draw-routine bekend zal zijn in alle van SHAPE afgeleide classes. Deferred classes in Eiffel zijn vergelijkbaar met abstracte basis-classes in C++, waar dit aangegeven wordt met het voorkomen van minstens één puur virtuele member-functie.

Multiple en repeated inheritance

Multiple inheritance betekent dat een class meer dan één class als basis-class heeft. In het volgende voorbeeld erft de class C zowel van A als van B:

```
class A
  — ... Eigenschappen van A
end;

class B
  — ... Eigenschappen van B
end;

class C
  inherit
    A;
    B
  — ... Toegevoegde eigenschappen
end  — C
```

Indien indirect tweemaal van één basis-class wordt geërfd, is er sprake van repeated inheritance. Meyer geeft zelf in zijn boek 'Object-Oriented Software Construction' het voorbeeld van een class DRIVER, en daarvan afgeleide classes US_DRIVER en FRENCH_DRIVER. Indien dan een nieuwe class FRENCH_US_DRIVER wordt gedefinieerd als afgeleide van FRENCH_DRIVER en US_DRIVER, zal de basis-class DRIVER tweemaal voorkomen in FRENCH_US_DRIVER. Gelukkig voorziet Eiffel in mogelijkheden om basis-members te delen of onder een andere naam op te nemen, iets wat in combinatie niet mogelijk is in C++.

Genericity

Een ander zeer belangrijk onderdeel van Eiffel is genericity. Dit houdt in dat een class wordt gedefinieerd voor een onbepaald datatype T, waarvoor op een later tijdstip elk gewenst datatype kan worden ingevuld. Zo

kan een generieke ARRAY-class worden gedefinieerd, waarna eenvoudig ARRAY's van integers, strings of door de gebruiker gedefinieerde typen kunnen worden gedefinieerd. Dit maakt de complete herdefinitie van totaal nieuwe classes INT_ARRAY, STRING_ARRAY en dergelijke overbodig, en kan dus veel tijd besparen. Samen met inheritance kan genericity bijdragen tot een hoge graad van software-hergebruik.

Een generieke STACK-class wordt in Eiffel als volgt gedefinieerd:

```
class STACK [Type]
  export
    is_empty, push, pop
  feature
    is_empty : BOOLEAN is
      do
        -- Controleer of stack leeg is
      end;
    push( x : Type ) is
      do
        -- Voeg element toe aan de stack
      end;
    pop : Type is
      do
        -- Geef bovenste element terug
      end;
end -- STACK
```

In deze definitie is Type een algemeen type, waarvoor een simpel type (bijvoorbeeld INTEGER) maar ook een class-type (bijvoorbeeld POINT) kan worden ingevuld. Het volgende fragment definieert STACK's van INTEGER's en van POINT's:

```
p_stack : STACK[ POINT ];
i_stack : STACK[ INTEGER ];
```

Nu kan p_stack alleen POINT's bevatten, en i_stack alleen INTEGER's. Indien een stack gewenst is met gemixte elementen, moet inheritance gebruikt worden.

Condities, assertions en exceptions

Zoals reeds gezegd is Eiffel sterk gericht op correctheid en robuustheid van applicaties. Door middel van pre- en postcondities, assertions en exception handling kan een programmeur zijn code een stuk betrouwbaarder maken. Een belangrijk concept hierbij is 'programming by contract': een class en zijn gebruikers sluiten impliciet een soort contract waarin de rechten en plichten van beiden zijn vermeld. Meestal komt het hierop neer: als de gebruiker van een class een member-routine aanroept onder de juiste omstandigheden dan zorgt de class ervoor dat de juiste resultaten worden afgeleverd.

Zo kan de push-routine van een STACK-class alleen aangeroepen worden indien de STACK niet vol is, en de pop-routine indien de STACK niet leeg is. Indien hieraan wordt voldaan, zullen de routines de juiste resultaten opleveren. Deze zogenaamde pre-condities kunnen in Eiffel met behulp van de require-clause worden aangegeven, waarbij we als voorbeeld een generieke STACK-class nemen:

```
class STACK [Type]
  -- ... export, features
  push( x : Type ) is
    require
      not is_full
    do
      -- Voeg element toe aan de stack
    end;

  pop : Type is
    require
      not is_empty
    do
      -- Haal bovenste element van de
stack
    end;

end  -- STACK
```

We zien hier duidelijk de condities waaraan de huidige STACK moet voldoen, willen de push- en pop-operaties slagen. Overigens doen we hier de Eiffel-conventie, dat alleen procedures side-effects kunnen hebben, geweld aan doordat de pop-functie de toestand van de STACK verandert.

Post-condities kunnen worden gebruikt om aan te geven waaraan de toestand van de class moet voldoen als de operatie uitgevoerd is. Zo kan een STACK nooit leeg zijn indien net een push-operatie is uitgevoerd, en nooit vol indien een pop-operatie is uitgevoerd. Post-condities kunnen worden aangegeven met behulp van een ensure-clause:

```
class STACK
  -- ... export, features
  push( x : Type ) is
```

```
    require
      not is_full
    do
      — Voeg element toe aan de stack
    ensure
      not is_empty
    end;

  pop : Type is
    require
      not is_empty
    do
      — Haal bovenste element van de
stack
    ensure
      not is_full
    end;

end  — STACK
```

Doordat de pre-conditie controleert op de juiste toestand van een STACK, hoeft de code na de 'do' geen rekening te houden met een onjuiste toestand van het object.

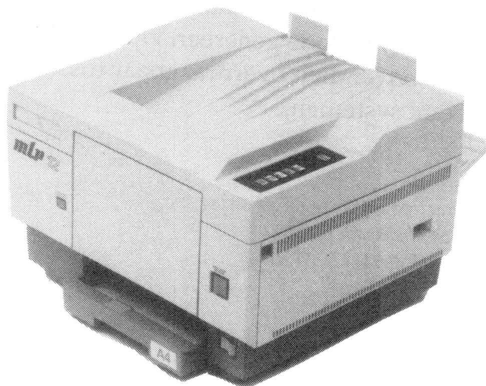
Naast pre- en post-condities kunnen classes in Eiffel ook variant- en invariant-clauses hebben. Bovendien is een aantal andere keywords beschikbaar voor exception handling, waar we hier verder niet op zullen ingaan. Al deze eigenschappen zorgen ervoor dat er zeer betrouwbare software geschreven kan worden in Eiffel.

Concluderend kan worden gesteld dat Eiffel een zeer veelzijdige en uitgebreide object-georiënteerde taal is, waar betrouwbaarheid hoog in het vaandel staat. Doordat de Eiffel-translator C-code genereert zijn Eiffel-applicaties bovendien overdraagbaar tussen de meeste systemen.

Informatie:
Interactive Software Engineering Inc.,
270 Storke Road, Suite 7
Goleta, CA 93117 - USA

Programmeren in PostScript

PostScript is een bekende pagina-opmaaktaal om laserprinters mee aan te sturen. Het is door Adobe Systems ontworpen om de kloof te overbruggen tussen computergestuurde opmaaksystemen en raster-uitvoersystemen. In dit artikel zullen we ingaan op de achtergronden van PostScript en de mogelijkheid om zelf printers in PostScript aan te sturen.



Veel programma's zijn in staat uitvoer in PostScript-formaat af te leveren. Weinigen weten echter dat PostScript een complete programmeertaal is, waarin ook zelf 'geprogrammeerd' kan worden. PostScript-bestanden hebben een normaal ASCII-formaat, en kunnen dus in een editor worden bijgewerkt. PostScript-uitvoer van opmaakprogramma's kan zodoende handmatig bijgesteld worden; ook kunnen er stand-alone 'toepassingen' in PostScript worden geschreven.

Echte PostScript-printers bevatten een ingebouwde hardware-interpreter en zijn tot op heden redelijk prijzig. Een goedkopere oplossing is het gebruiken van software PostScript-interpreters, zoals Freedom of Press of GoScript. Deze programma's kunnen een PostScript-programma uitvoeren naar een reeks van printers. Met een inkjet-printer kunnen heel behoorlijke resultaten worden bereikt als gebruik wordt gemaakt van zo'n software-interpreter. Voor het heavy-duty werk is echter een printer nodig waarbij de PostScript-ondersteuning vanuit de hardware komt.

Onafhankelijkheid van de printer

De PostScript-taal is device-onafhankelijk, dat wil zeggen dat een PostScript-programma geen specifieke commando's en codes bevat voor een bepaalde printer. In plaats hiervan worden pagina's in abstracte grafische objecten beschreven, terwijl de PostScript-interpreter zelf het 'zware' werk verricht. Deze interpreter zet de PostScript-programma's om in de stuurcodes van de desbetreffende printer. Doordat PostScript onafhankelijk is van een specifieke printer en PostScript-programma's in het ASCII-formaat staan, is PostScript een perfecte taal om compleet opgemaakte documenten electronisch te verzenden. Zo is het bij universiteiten gebruikelijk dat een bepaald wetenschappelijk document zowel in 'plain text' als in PostScript-formaat op een systeem wordt gezet. Een andere universiteit, die aan het netwerk verbonden is, kan dan het PostScript-bestand downloaden en heeft na het uitdraaien een compleet gelayout boekwerkje. Een nadeel is vaak dat de omvang van een PostScript-bestand enorm kan worden; 600 kiloByte voor een document van 65 pagina's is niets bijzonders in de PostScript-wereld. We hebben zelf eens een geconverteerde .PCX-afbeelding van 4,5 MegaByte gehad.

De structuur van de taal

Zoals reeds vermeld, wordt in PostScript een opgemaakte pagina beschreven in grafische objecten. Een letter is een voorbeeld van een object, maar ook ingebouwde of nieuw te definiëren figuren. PostScript biedt de mogelijkheid om vele bewerkingen te verrichten op objecten, zoals buigen, van schaduw voorzien, van grootte veranderen enzovoorts.

Een bijzondere eigenschap van PostScript is dat de taal gebruik maakt van postfix-notatie. Dit houdt in dat eerst de argumenten en dan pas een commando wordt gegeven, x y moveto in plaats van moveto(x,y). Bij het verwerken van postfix-expressies wordt gebruik

gemaakt van een stack, waarvoor een aantal operatoren beschikbaar is.

Een eenvoudig programma

We zullen nu het welbekende 'Hello, world!'-programma in PostScript geven. Commentaren worden in PostScript ingeluid met een procent-teken (%), waarna ze gelden tot het einde van de regel.

```
% Hello.PS
% Hello, world in PostScript

/Courier findfont 12 scalefont setfont
100 400 moveto
(Hello, world!) show
showpage
```

Het eerste commando, /Courier findfont, zorgt ervoor dat de interpreter het Courier font selecteert. Dit is een van de vier fonts die elke PostScript-implementatie ondersteunt. De andere drie zijn: Times-Roman, Helvetica en Symbol, die dus ook in plaats van het 'Courier' hadden kunnen staan. Hierbij bevat 'Symbol' een aantal mathematische en andere speciale symbolen. Met scalefont wordt het font 'geschaald' naar een bepaalde omvang, in het bovenstaande voorbeeld een 12-punts letter. De setfont operator tenslotte maakt het geselecteerde font actief.

Het commando moveto kan gebruikt worden om de huidige positie aan te geven. Tekst staat in PostScript tussen ronde haakjes, en wordt met het show-commando afgedrukt. Het showpage-commando tenslotte zorgt ervoor dat het geheel wordt uitgevoerd naar de printer.

Procedures

In PostScript kunnen procedures op de volgende manier worden gedefinieerd:

```
/myProc
  {
    % ... commando's ...
  } def
```

Nu zal myProc een procedure zijn met als body de commando's die tussen de accolades staan. Een ander voorbeeld van een procedure:

```
/cm
  {
    28 mul
  } def
```

Omdat PostScript-coördinaten ongeveer 1/28 van een centimeter zijn, worden deze met de procedure cm omgezet in centimeters. Voor de aanroep wordt het PostScript-coördinaat op de stack gezet (door het getal in te voeren), waarna 28 op de stack wordt gezet en de mul-instructie wordt aangeroepen. Deze popt de laatste twee getallen van de stack en vermenigvuldigt ze met elkaar. Het resultaat wordt wederom op de stack gezet. Het definiëren van de procedure cm betekent dus dat er eenvoudig, in plaats van met PostScript-coördinaten, met centimeters kan worden gewerkt. Het moveto-commando uit het voorbeeldprogramma zou als volgt gebruik kunnen maken van de cm-procedure:

```
4 cm 14 cm moveto
```

Graphics

Ongeveer 30% van alle PostScript-operatoren heeft betrekking op graphics. Het lineto-commando kan gebruikt worden om lijnen te tekenen, gerekend vanaf de huidige positie. Het volgende PostScript-programma laat zien hoe lijnen getekend kunnen worden.

```
% Line.PS

/cm
  {
    28 mul
  } def

5 cm 15 cm moveto      % gaat naar (5,15)
10 cm 15 cm lineto     % lijn tot (10,15)
8 cm 4 cm lineto       % lijn tot (8,4)
showpage
```

Natuurlijk biedt PostScript veel meer dan de enkele mogelijkheden die we in dit artikel hebben behandeld. In de volgende DOS Special zullen we een PostScript-interface bouwen waarmee we vanuit een hogere programmeertaal PostScript-printers kunnen aansturen. Voor diegenen die niet zolang willen wachten, is er een beperkt aantal goede PostScript-boeken verkrijgbaar in de boekhandel.

De Paradox Engine

Een techniek die bij Borland hoog in het vaandel staat, is interoperability. Dit houdt in dat Borland-programma's onderling zo compatible mogelijk zijn, bijvoorbeeld wat betreft bestandsformaten die verwerkt kunnen worden. Een mooi voorbeeld hiervan is ObjectVision, waarmee Paradox-tabellen verwerkt kunnen worden. Ook vanuit Quattro Pro, Borlands spreadsheet, kunnen deze bestanden gebruikt worden.

De interoperability van veel produkten is gebaseerd op een stuk software, de Paradox Engine geheten. Dit is een bibliotheek van routines waarmee Paradox-tabellen (de Paradox .DB-bestanden) kunnen worden verwerkt. Van de library is kort geleden versie 2.0 uitgebracht, die met Pascal, C en C++ gebruikt kan worden.

Het Paradox-programma

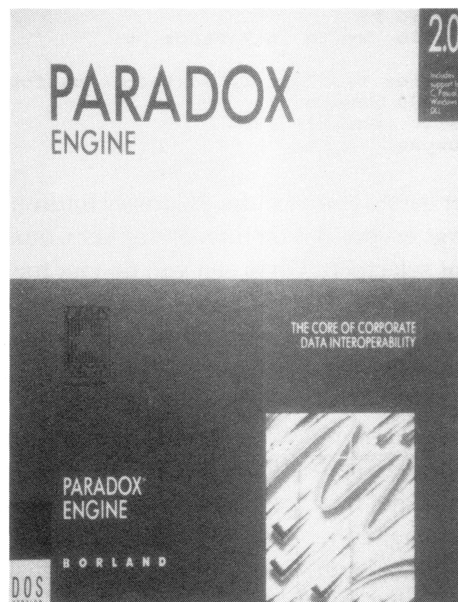
Paradox is de relationele database van Borland, en is een zeer krachtig programma dat ook geschikt is voor multi-user omgevingen. Query by Example ofwel QBE wordt door Paradox ondersteund, en met behulp van de Paradox SQL Link kan het programma gebruikt worden om gegevens van een SQL database server op te vragen. In Paradox ingebouwd is de Paradox Application Language (PAL) en de Paradox Personal Programmer, een applicatiegenerator.

```
View  Ask  Report  Create  Modify  Image  Forms  Tools  Scripts  Help  Exit
Output, design, or change a report specification.
```

SAMPLE	Customer Id	Last Name	Init	Street
1	1386	Aberdeen	F	45 Utah Street
2	1784	McDougal	L	4950 Pullman Ave NE
3	2579	Chavez	L	Cypress Drive
4	5720	Helms	D	52 Brattle Street
5	8996	Smith	J	Hotel Americain
6	9650	O'Hare	C	1 Airport Drive

State	Zip	Country	Credit
DC	20032		50,000.00
WA	98105		1,250,000.00
FL	32938		250,000.00
MA	02138		85,000.00
		France	1,000,000.00
IL	60542		2,500,000.00

Het Paradox database-programma



De Paradox Engine

Om Paradox data-bestanden vanuit C of Pascal-programma's te kunnen gebruiken, is de Paradox Engine beschikbaar. Naast toegang tot de Paradox-bestanden biedt de Engine ook ondersteuning van netwerk-databases. Met de meer dan 70 routines die de Engine bevat kunnen programma's geschreven worden die de kracht van Paradox combineren met de kracht van een programmeertaal als C of Pascal. Zodoende kunnen programma's worden geschreven die niet met de ingebouwde PAL-taal mogelijk zijn. Tevens kan de Engine gebruikt worden om programma's als 'extra' het Paradox .DB-formaat te laten ondersteunen. Veel programma's bieden immers tegenwoordig de mogelijkheid om bijvoorbeeld dBase-bestanden te lezen of te schrijven.

We zullen eerst kort de opbouw van een Paradox database beschrijven. Zoals in de fi-

Een Windows-toepassing met de Paradox Engine

guur te zien is bestaat een Paradox datafile uit rijen en kolommen. Zo'n datafile heeft de extensie .DB en wordt in Paradox-jargon Table genoemd. De kolommen heten velden, de rijen records.(Zie illustratie vorige pagina)

Voordat de Engine-routines kunnen worden gebruikt, moet eerst de functie PXInit (PXNetInit voor een netwerk) worden aangeroepen. Dit zorgt ervoor dat de Engine op de juiste manier wordt geïnitialiseerd. Vervolgens kan met PXTblOpen een tabel worden geopend, en kan de veld-informatie worden opgehaald met de PXFld... functies.

Het werken met de Paradox Engine is erg eenvoudig. Omdat de functies het zware werk verrichten en het hoofdprogramma niets te maken heeft met bestandsstructuren, indices en dergelijke, kan snel een werkende applicatie worden opgezet. Hoewel zowel Pascal als C++ van Borland object-georiënteerde talen zijn, bevat de Paradox Engine nog geen object-georiënteerde schil. Borland kennende zal dit echter niet lang op zich laten wachten. De huidige versie van de Engine is geschikt voor Turbo C 2.0 (dat overigens niet meer geleverd wordt), Turbo C++, Borland C++ en Turbo Pascal. Een DLL (dynamic link library) voor gebruik onder Windows wordt eveneens standaard bijgeleverd.

Concluderend kan worden gesteld dat de Paradox Engine een goede oplossing is voor problemen die niet vanuit Paradox zelf kunnen worden opgelost, zoals het inlezen van gegevens via de seriële poort of andere 'low-level' aangelegenheden. Bovendien biedt de Engine de mogelijkheid om eenvoudig en snel ondersteuning van Paradox-tabellen in bestaande Pascal- of C- programma's in te bouwen.

Manipuleren van de DOS printer queue

Het MS-Dos programma PRINT.COM is een geheugen-residente print-spooler waarmee in de achtergrond bestanden kunnen worden afgedrukt. Het PRINT-programma bevat hiervoor een queue waarin alle namen van bestanden die nog moeten worden afgedrukt, opgesomd staan. Normaal gesproken is deze queue alleen via het PRINT-programma beschikbaar, maar met de sources die we in dit artikel zullen geven kan de queue ook vanuit uw eigen programma's worden gemanipuleerd. Tevens bieden de functies een aantal opties die niet met PRINT mogelijk zijn, zoals het toevoegen van een bestand op een willekeurige plaats in de queue. Het voorbeeldprogramma bevat een aantal van deze mogelijkheden.

PRINT [/D:device] [/B:size] [/U:ticks1] [/M:ticks2] [/S:ticks3] [/Q:qsize] [/T] [[drive:][path]filename[...]] [/C] [/P]

/D:device	Specificeert het uitvoerapparaat (bijvoorbeeld PRINT /D:LPT1 of /D:COM1).
/B:size	Stelt de omvang van de PRINT-buffer in (in bytes). De default-waarde is 512 Bytes, de maximum-waarde 16 kiloByte.
/U:ticks1	Wacht 'ticks1' kloktikken voordat de printer beschikbaar is. Normaal gesproken wordt één kloktik gewacht.
/M:ticks2	Geeft aan hoeveel kloktikken het kost om een teken af te drukken. Indien deze optie niet wordt gebruikt, wordt een waarde van 2 tikken gebruikt.
/S:ticks3	Geeft telkens 'ticks3' achtereenvolgende tikken aan het PRINT-programma. De verstekwaarde hiervan is acht tikken per keer.
/Q:qsize	Geeft aan hoeveel bestanden er maximaal in de PRINT-queue kunnen staan. De standaard waarde bedraagt tien bestanden en mag maximaal 32 bedragen.
/T	Verwijdert alle bestanden uit de print-queue.
/C	Verwijdert het aangegeven bestand en alle bestanden die daarop volgen uit de print-queue.
/P	Voegt het aangegeven bestand en alle volgende bestanden toe aan de PRINT-queue.

Gebruik van het Print-commando

De PRINT- queue

Een queue is te vergelijken met een wachtrij voor een kassa in de winkel; wie het eerst aankomt wordt het eerst geholpen. Het PRINT-commando bevat een ingebouwde queue waarvan de grootte kan worden ingesteld, die standaard 10 bestanden kan bevatten. We zullen verderop laten zien hoe de queue is opgebouwd en hoe we de inhoud kunnen wijzigen. Dit kan bijvoorbeeld gebeuren in toepassingsprogramma's, die dan gebruik kunnen maken van de diensten van het PRINT-programma, zodat een ingebouwde printer-spooler ontstaat.

Het PRINT-programma

PRINT is het enige multitasking DOS-programma, dat na het laden permanent in het geheugen verblijft, en wordt sinds versie 2 van MS-DOS geleverd. Als PRINT wordt gebruikt om bestanden af te drukken kan ondertussen worden doorgewerkt en zo ontstaat dus de illusie dat twee programma's tegelijkertijd draaien. In werkelijkheid koppelt PRINT zichzelf aan de timer-interrupt, die ongeveer 18 maal per seconde wordt aangeroepen door het systeem. Bij elke aanroep wordt weer een klein stukje van de tekst naar de printer gestuurd. We zullen hier niet verder ingaan op de techniek die dit mogelijk maakt, maar kijken hoe we vanuit onze eigen programma's met PRINT communiceren, hetgeen vanaf DOS 3 mogelijk is.

Het intypen van PRINT zonder argumenten zal de inhoud van de huidige queue laten zien.
Enkele functies kunnen niet met PRINT worden verricht; bijvoorbeeld een bestand tussenvoegen aan de queue, of een bestand in de queue dupliceren. Dit is wel mogelijk met enig programmeerwerk, zoals we zullen laten zien.

De multiplex-interrupt

Alle PRINT-queue functies die we in dit artikel zullen geven maken gebruik van de multiplex-interrupt met de hexadecimale code 2F. Een multiplex-interrupt is bedoeld om toegang te bieden tot meerdere sets interrupt-routines. Voor PRINT.COM zijn functies 0 en 1 (deze waarden worden voor het aanroepen van int 0x2F in het AH-register geplaatst). De meeste subfuncties maken onderdeel uit van functie 1.

De eerste functie die we aanroepen is AX = 0x100, dat wil zeggen subfunctie 0 (in AL) van de functie 1 (in AH). Het AL-register vertelt ons daarna meer over de installatie van PRINT:

AL = 0	*Niet geïnstalleerd, maar kan geïnstalleerd worden.*
AL = 1	*Niet geïnstalleerd en kan niet geïnstalleerd worden.*
AL = 0xFF	*Geïnstalleerd.*

Het afdrukken van een bestand gaat met subfunctie 1 van functie 1; in AX wordt derhalve de waarde 0x101 geplaatst. Bovendien moet DS:DX het adres bevatten van het zogenaamde submit-pakket, dat de volgende structuur heeft:

```
struct SubmitPack
{
  unsigned char Level;   // Altijd nul
  char far *FileName;
};
```

De waarde van Level moet altijd nul bedragen en FileName is een far pointer naar een bestandsnaam. Indien het AX-register na terugkeer een waarde lager dan 16 bevat, is er ergens een fout opgetreden. Dit kan bijvoorbeeld het geval zijn als de buffer vol is.
Subfunctie 2 van functie 1 (AX = 0x102) kan gebruikt worden om een of meer bestanden uit de queue te verwijderen. Hierbij wordt DS:DX gevuld met het adres van de string die de bestandsspecificatie bevat, waarin ook wildcards mogen worden gebruikt. Ook hier geeft een waarde van AX die kleiner dan 16 is na terugkeer van de functie aan dat er ergens iets fout is gegaan. Met subfunctie 3 van functie 1 wordt de gehele queue geleegd.

Een belangrijke functie is 'Get status', subfunctie 4 van functie 1. Hiermee wordt de huidige status van PRINT opgevraagd, maar wordt als neveneffect het afdrukken stopgezet. De volgende foutmeldingen kunnen in het AX-register voorkomen:

0x02	*Bestand niet gevonden*
0x03	*Pad niet gevonden*
0x04	*Teveel bestanden geopend*
0x05	*Toegang geweigerd*
0x08	*Queue is vol*
0x09	*Spooler is bezet*
0x0C	*Te lange bestandsspecificatie*
0x0F	*Ongeldige drive-aanduiding*

Bij terugkeer bevat DS:SI het adres van de PRINT-queue buffer in het geheugen.

Iedere bestandsnaam in de queue beslaat exact 64 Bytes (maar wordt eventueel eerder afgesloten met een null-teken), terwijl het einde van de queue wordt aangegeven door een string die begint met het null-teken. In diverse functies is te zien hoe de queue gemanipuleerd kan worden.

Resumerend kan subfunctie 4 dus voor drie doeleinden gebruikt worden: het opvragen van de status, het opvragen van het bufferadres in het geheugen en het stopzetten van het afdrukproces.
Het opnieuw starten van het afdrukken kan gebeuren met subfunctie 5 van functie 1, waarbij dus AX de waarde 0x105 bevat. Een

`Print/Append`	Voegt een bestand toe onderaan de queue
`Cancel`	Verwijdert een bestand uit de queue
`Clear`	Maakt de buffer leeg
`Stop`	Stopt het afdrukken tot nader order
`Start`	Hervat het afdrukken
`Show`	Laat de inhoud van de queue op het beeldscherm genummerd zien
`Index`	Geeft de index van een bestand in de queue
`GetFile`	Geeft de bestandsnaam van een element in de queue
`Duplicate`	Dupliceert een bestandsnaam in de queue
`Insert`	Voegt een bestand toe op een willekeurige plaats in de queue
`nFiles`	Geeft het huidige aantal bestanden in de queue
`IsEmpty`	Controleert of de queue leeg is

Interface van de PrintQ class

C-functie die dit bewerkstelligt zou er als volgt kunnen uitzien:

```
void PrintQueueStart()
{
  union REGS r;

  r.x.ax = 0x105;
  int86( 0x2F, &r, &r );
}
```

Nu we alle basisfuncties behandeld hebben, kunnen we ingaan op de implementatie van de sources. Het is duidelijk dat niet alles wat we willen met standaard functies gedaan kan worden, en we zullen dus sommige functies (zoals Insert) op een andere manier moeten schrijven.

De implementatie

De sources zijn geschreven in C++, waarvoor we de Borland C++ compiler gebruikten. De class PrintQ zorgt voor de inkapseling van alle functies en data die betrekking hebben op de printer queue. Van deze class wordt typisch slechts een object geïnstantieerd, die vervolgens als een printer queue aangesproken kan worden. We hebben C++ in dit geval puur voor de encapsulation gebruikt; de member functies en de data members kunnen echter ook in regulier C worden geschreven als globale functies en variabelen. De class is als gewoonlijk onderverdeeld in twee delen: een private en een public deel. Het eerste deel bevat alle variabelen en functies die we alleen intern gebruiken en wordt het implementatie-gedeelte genoemd. Programma's die de PrintQ class gebruiken hebben niets te maken met deze gegevens. Het public gedeelte bevat de functies waarmee we gaan werken en wordt ook wel het interface-gedeelte genoemd.

Enkele functies zijn private, omdat ze alleen binnen de PrintQ zelf gebruikt worden. Een voorbeeld is CheckDos, een functie die controleert of DOS de juiste versie heeft. Deze versie moet hoger dan 3 zijn, omdat eerdere versies geen PRINT functies in de multiplex-interrupt ondersteunden.

De meeste functies zijn vrij eenvoudig en doen weinig meer dan het aanroepen van de onderliggende functie van de multiplex-interrupt. Vermeldenswaardig is de manier waarop argumenten in de registers worden geplaatst; hiertoe kennen we een pointer-waarde toe aan het DX-register, geconverteerd naar een unsigned integer. Een kenmerk van C++ is dat casts (conversies) genoteerd kunnen worden als functie-aanroepen, vandaar unsigned(FileName) in plaats van (unsigned)FileName. De PrintRequest-member geeft weer hoe een bestand 'gesubmit' wordt naar de interrupt. Belangrijk is hierbij dat de pointer naar de bestandsnaam van het type 'far' is.

Voor enkele functies is wat meer werk vereist. Hebben we tot op heden nog niet direct de buffer in het geheugen aangesproken, met functies als Show en Insert zijn we gedwongen dit wel te doen. Hiertoe bevat de PrintQ een data member QueuePtr, die bij initialisatie van het PrintQ object het adres van de buffer in het geheugen zal gaan bevatten. Vervolgens gaan we met een lokale pointer de buffer af. We stoppen waar het eerste teken nul bedraagt.

De Insert-functie maakt gebruik van de private functie nFilesNoStart, die het aantal bestanden teruggeeft maar het afdrukken niet hervat. Dit aantal hebben we nodig om te controleren of de gebruiker een juiste index heeft meegegeven. Om een bestand in de queue te plaatsen, moeten eerst de achterliggende bestanden in de queue een plaats worden verschoven. Hiervoor gebruiken we de _fmemcpy- en _fstrncpy-functies, Borland-speficieke functies om far data te kopiëren. Als laatste wordt een afsluitende nul gezet om het einde van de queue aan te geven.

Het gebruik van de PrintQ class

Nu alle functies beschreven zijn, kunnen we laten zien hoe de class gebruikt kan worden. In een main-programma wordt één object aangemaakt van het type PrintQ. De PrintQ-constructor zorgt ervoor dat automatisch alle variabelen hun juiste waarden krijgen. Vervolgens kunnen de member-functies gebruikt worden om operaties te verrichten op de printer-queue. Het volgende programma voegt een bestandsnaam toe aan de top van de queue:

```
int main()
{
  PrintQ thePrintQ;

  thePrintQ.Print( "theFile.ps" );
}
```

```
// PrintQ.h
// Definitie van de PrintQ class

#if !defined PRINTQ_H
#define PRINTQ_H

class PrintQ
{
  private:
    static int nInstances;

      // Structure of print request
    struct
    {
      char Level;
      char far *FileName;
    } PrintRequest;

      // Pointer to Print queue in memory
    char far *QueuePtr;

      // Private member functions
    void CheckDos();
    int Status();
    void Error( char *Fmt =0, ... );
    int nFilesNoStart();

  public:
    PrintQ();
    ~PrintQ();

      // Check if PRINT.COM installed
    int IsInstalled();

      // Druk een of meer bestanden af
    void Print( char far *FileName, int n =1 );
      // Synoniem: Append
    void Append( char far *FileName, int n =1 )
    {
      Print( FileName, n );
    }

      // Verwijder een of meer bestanden uit de
      // print queue (wildcards toegestaan)
    void Cancel( char *FileName );

      // Maak de queue leeg
    void Clear( );

      // Stop het afdrukken tijdelijk
    void Stop();

      // Hervat het afdrukken
    void Start();

      // Laat de inhoud van de queue zien
    void Show();

      // Retourneer de index van een bestand
    int Index( char *FileName );

      // Dupliceer een bestand in de queue
    void Duplicate( char *FileName, int n =1 );

      // Voeg een bestand toe aan de queue
    void Insert( char *FileName,
                 unsigned int At =1 );

      // Retourneer het aantal bestanden
      // in de queue
    int nFiles();

      // Controleer of de queue geen
      // bestanden meer bevat
    int IsEmpty();

      // Retourneer bestandsnaam van bestand i
    char *GetFile( int i );
};

#endif
```

```
// PrintQ.cpp

#include "PrintQ.h"

#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <conio.h>
#include <mem.h>

enum
{
  FileLength = 64,
  Multiplex = 0x2F
};

int PrintQ::nInstances = 0;

PrintQ::PrintQ()
{
  if ( nInstances == 0 )
  {
    nInstances++;
  }
  else
  {
    Error( "PrintQ already installed" );
    exit( 1 );
  }
  if ( !IsInstalled() )
    Error( "PRINT.COM not loaded" );
  PrintRequest.Level = 0;

  union REGS r;
  struct SREGS s;

  r.x.ax = 0x104;
  int86x( Multiplex, &r, &r, &s );

  QueuePtr = (char far *)MK_FP( s.ds, r.x.si );
}

PrintQ::~PrintQ()
{
  nInstances—;
}

int PrintQ::Status()
{
  CheckDos();
  union REGS r;

  r.x.ax = 0x100;
  int86( Multiplex, &r, &r );

  return r.h.al;
}

void PrintQ::Error( char *Fmt, ... )
{

  printf( "\nPrintQ Error: ");
  if ( Fmt )
  {
    va_list ap;
    va_start( ap, Fmt );
    vprintf( Fmt, ap );
    va_end( ap );
  }
  else
    printf("General failure");

  exit( 1 );
}

void PrintQ::CheckDos()
{
  if ( _osmajor < 3 )
    Error( "Must have DOS 3 or higher" );
}

int PrintQ::IsInstalled()
{
  return Status() == 0xFF;
}

void PrintQ::Cancel( char *FileName )
{
  union REGS r;
  r.x.dx = unsigned( FileName );
  r.x.ax = 0x102;
  int86( Multiplex, &r, &r );

  if ( r.x.ax <= 15 )
    Error( "Could not cancel %s", FileName );
}

void PrintQ::Print( char far *FileName, int n )
{
  PrintRequest.FileName = FileName;
  union REGS r;

  r.x.dx = unsigned( &PrintRequest );
  r.x.ax = 0x101;
  int86( Multiplex, &r, &r );

  if ( r.x.ax <= 15 )
    Error( "Could not print %s", FileName );

  if ( n > 0 )
    Print( FileName, —n );
}

void PrintQ::Clear()
{
  union REGS r;

  r.x.ax = 0x103;
  int86( Multiplex, &r, &r );
}

void PrintQ::Stop()
{
  union REGS r;

  r.x.ax = 0x104;
  int86( Multiplex, &r, &r );
}

void PrintQ::Start()
{
  union REGS r;

  r.x.ax = 0x105;
  int86( Multiplex, &r, &r );
}

void PrintQ::Show()
{
  int FileCount = 0;
  volatile char far *qPtr = QueuePtr;

  printf( "Queue located at %Fp\n\n", QueuePtr );

  if ( IsEmpty() )
  {
    printf( "Print queue is empty\n" );
    return;
  }
  else
    printf( "%i files in print queue:\n\n",
    nFiles() );

  Stop();

  while ( *qPtr )
  {
    printf( "%2i: ", FileCount );
```

```
    for ( int i=0; i<FileLength && qPtr[i]; i++ )
      putchar( qPtr[i] );
    puts( "" );
    qPtr += FileLength;
    FileCount++;
  }

  Start();
}

  // nFilesNoStart
  // Geef het aantal bestanden in de queue,
  // zonder dat het printen weer hervat wordt.
int PrintQ::nFilesNoStart()
{
  Stop();
  char far *qPtr = QueuePtr;
  int Result = 0;

  while ( *qPtr )
  {
    Result++;
    qPtr += FileLength;
  }
  return Result;
}

int PrintQ::Index( char *FileName )
{
  Stop();
  char far *qPtr = QueuePtr;
  int Result, Found;

  Result = Found = 0;
  while ( *qPtr && !Found )
  {
    Found = _fstricmp( qPtr,
       (char far*)FileName )==0;
    Result++;
    qPtr += FileLength;
  }
  return Found ? Result-1 : -1;
  Start();
}

int PrintQ::nFiles()
{
  int Result = nFilesNoStart();
  Start();
  return Result;
}

int PrintQ::IsEmpty()
{
  return *QueuePtr == 0;
}

void PrintQ::Insert( char *FileName, unsigned At
)
{
  if ( At == 0 )
    Error( "Illegal attempt to enqueue file" );

  int Count = nFilesNoStart();

  if ( Count <= At )
    Print( FileName );
  else
  {
      // Plaats bestandsnaam in queue
    char far *Source = QueuePtr +
       FileLength * (Count-1);
    char far *Dest   = QueuePtr +
       FileLength * Count;

    for ( int i=Count; i > At; i-- )
    {
      _fmemcpy( (void far*)Dest,
                (void far*)Source, FileLength );
      Dest -= FileLength;
      Source -= FileLength;
    }

    _fstrncpy( QueuePtr + At*FileLength,
       (char far*)FileName, 63 );
    *( QueuePtr + (Count+1)*FileLength ) = 0;
  }
}

char *PrintQ::GetFile( int i )
{
  static char Result[FileLength];

  Result[0] = 0;

  if ( i < nFiles() )
    _fstrcpy( (char far*)Result,
      QueuePtr + i*FileLength );

  return Result;
}

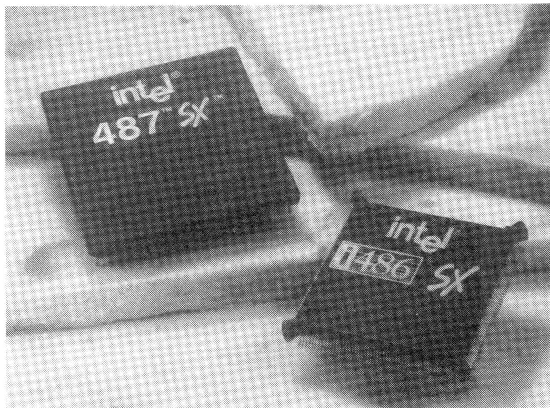
void PrintQ::Duplicate( char *FileName, int n )
{
  int Idx = Index( FileName ) + 1;

  for ( int i=0; i<n; i++ )
     Insert( FileName, Idx );
}

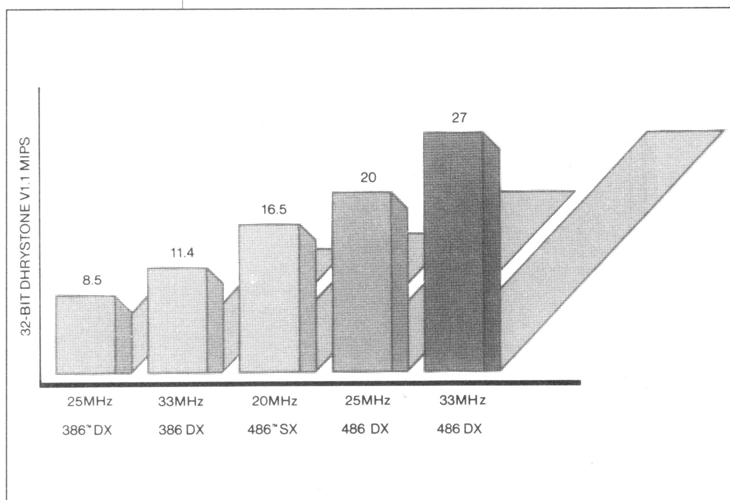

// eof PrintQ.cpp
```

De Intel 486 SX processor

Sinds enige tijd heeft Intel ook voor de 80486-processor een instapmodel, de 80486 SX. Deze processor mist enkele eigenschappen ten opzichte van de conventionele 486 (ook wel 486 DX genoemd) en heeft een iets lagere snelheid. Toch is deze snelheid nog steeds hoger dan de 386 DX-processor op 33 MHz; ter illustratie, de laatste processor zou ongeveer op 45 MHz moeten draaien om de 486 SX te kunnen bijbenen. We zullen iets dieper ingaan op de mogelijkheden die deze nieuwe chip ons verder te bieden heeft.



Sinds de 80486 DX uitgebracht werd, staat deze bekend als een hoge-snelheidsprocessor waarvan de prijs echter aan de hoge kant was. Naast een ingebouwde cache bevat deze processor een mathematische coprocessor, alsmede een memory management unit (MMU) met paging-capaciteiten. Momenteel is de 80486 verkrijgbaar in de kloksnelheden 25 en 33 MHz.



Net als destijds met de 386 DX werd door Intel nu ook besloten een goedkopere, gestripte versie van de 486 uit te brengen onder de naam 486 SX. De grootste verschillen met zijn grote broer zijn het gemis van een ingebouwde coprocessor en de lagere kloksnelheid van 20 MHz. Door het weglaten van de coprocessor is een 486 SX bereikbaar geworden voor een groter publiek. Niet iedere gebruiker heeft immers een even grote behoefte aan een mathematische coprocessor. Indien een 486 SX-computer bijvoorbeeld voornamelijk voor databasing wordt gebruikt zal de toegevoegde coprocessor weinig extra performance opleveren. Toch is de de 486 SX een echte 32-bits processor en is de interne structuur van de hoofdprocessor hetzelfde als die van de 486 DX, inclusief integer unit, geïntegreerde MMU en de cache. De integer unit maakt gebruik van RISC-ontwerptechnieken, waardoor kern integer-instructies in slechts een clock cycle uitgevoerd kunnen worden. De geïntegreerde memory management unit maakt het mogelijk 64 teraByte virtueel geheugen te adresseren.

Uitbreidingen voor de 486 SX

Mocht u op een later tijdstip bedenken dat uw 486 SX het zonder coprocessor toch niet kan bolwerken, dan is de 487 SX-coprocessor beschikbaar. Deze heeft echter een adviesprijskaartje van f 1500 excl. BTW zodat de totaalprijs toch weer richting 486 DX kruipt. De 487 SX is overigens zo'n 70% sneller dan de 387/33 coprocessor.

Boeken over Clipper 5.0

Werkwijzer Clipper 5.0: Een beknopte cursus

In de 'Werkwijzer'-serie van Sybex verscheen enige tijd geleden dit boekje van de auteurs Berns en Heinz. Het is vooral gericht op mensen die in korte tijd met dit databaseprogramma willen leren werken, maar die al ervaring hebben in het programmeren in de dBase-taal. De bij Clipper horende hulpprogramma's zoals DBEDIT, RL en DBU worden besproken, evenals natuurlijk het gebruik van de compiler en de linker. Tevens wordt de debugger behandeld, en wordt een voorbeeldprogramma ontwikkeld voor de koppeling met WordPerfect-bestanden voor mailings.

Titel: Werkwijzer Clipper 5.0
Auteurs: Klaus Berns en Andreas Heinz
Taal: Nederlands
Uitgeverij: Sybex BV, Soest
Omvang: 168 pagina's
ISBN: 90-5160-322-3
Prijs: *f* 24,95

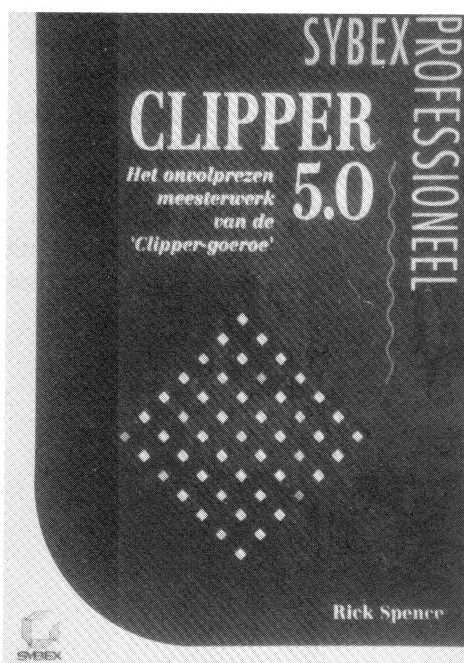
Werkwijzer Clipper 5.0: Functies en commando's

Twee werkwijzers over Clipper 5.0 vielen ons deze keer ten deel. Dit exemplaar is iets dunner dan zijn broertje en behandelt alle commando's, functies en opties van de hulpprogramma's. Het boekje is vooral bedoeld als 'quick reference' voor alle gebruikers van Clipper 5.0. Ook wordt de EXTEND-interface naar andere talen zoals C en assembly genoemd, alsmede de object-georiënteerde uitbreidingen zoals de error-class. In het boek worden geen voorbeeldprogramma's uitgewerkt, het is enkel een complete lijst van alle mogelijke Clipper-commando's en functies.

Titel: Werkwijzer Clipper 5.0
Auteurs: Frank Brakenhoff en Dré van Melis
Taal: Nederlands
Uitgeverij: Sybex BV, Soest
Omvang: 126 pagina's
ISBN: 90-5160-323-1
Prijs: *f* 24,95

Clipper 5.0 (Sybex professioneel)

Dit boek is een absolute must voor professionele Clipper-programmeurs. In bijna 800 pagina's worden alle eigenschappen van deze compiler uit de doeken gedaan, inclusief de koppeling met C, het gebruik van netwerken en de bestandsstructuren van de meestgebruikte bestanden. Het eerste hoofdstuk bespreekt het compileren en linken van Clipper-toepassingen, waarbij ook het gebruik van libraries en het RMAKE-programma worden behandeld. In het tweede hoofdstuk wordt de Clipper-taal uitgelegd, het gebruik van de preprocessor, lokale variabelen, codeblokken en de voorgedefinieerde klassen die deel uitmaken van de object-georiënteerde uitbreiding van Clipper 5.0. Hoofdstuk drie



is in zijn geheel gewijd aan arrays en de daarvoor beschikbare functies. Het vierde hoofdstuk behandelt het SET KEY TO-commando en scherm- en database-stacks. Memovelden en het werken met bestanden vanuit Clipper-programma's worden in de volgende twee hoofdstukken besproken. Een belangrijk onderdeel van Clipper-programma's, namelijk de gebruikersinterface, wordt in hoofdstuk zeven besproken. Hierbij worden ook de TbColumn en TBrowse classes behandeld; een vreemde term is hier 'gevalsvariabele'. Pas na enig zoekwerk werd het duidelijk dat hier member-variabelen van een klasse bedoeld werden, die voor objecten van die klasse beschikbaar zijn. In zo'n geval zou het beter zijn de Engelse term te gebruiken, als de vertaling onduidelijkheden oplevert. Het gebruiken van databases wordt in de bijna honderd pagina's van hoofdstuk acht, besproken. Daarna wordt de interface tussen Clipper en C besproken, een mogelijkheid die ook steeds meer mensen gebruiken om de 'low-level'-kracht van C met de databasekracht van Clipper te combineren. Na een hoofdstuk over het gebruik van Clipper in een netwerk wordt in het laatste hoofdstuk ingegaan op de bestandsstructuren van DBF, DBT, FRM, LBL, MEM, NDX en NTX-bestanden. Dit is vooral handig voor programmeurs die in een andere taal dan Clipper gebruik willen maken van bestanden in de genoemde formaten. De uitleg maakt gebruik van C-structuren, maar is zonder meer van toepassing op andere talen, zoals Pascal.

Eigenlijk heeft dit boek maar één nadeel: de index. Hoe kan de index van een boek van dit formaat nu maar één trefwoord beginnend met een 'J' bevatten, vier met een 'K' enzovoort. Voor de rest is Clipper 5.0 uit de Sybex Professioneel-serie is een zeer compleet boek voor de veeleisende Clipper-programmeur.

Titel: Clipper 5.0 (Sybex Professioneel)
Auteur: Rick Spence
Taal: Nederlands
Uitgeverij: Sybex BV, Soest
Omvang: 780 pagina's
ISBN: 90-5160-317-7
Prijs: *f* 99,--

Werken met Clipper 5.0

Hier vierde boek van Sybex over Clipper 5.0 is volgens de achterflap een handleiding, leerboek en naslagwerk voor zowel starters als gevorderden. Het boek bevat bovendien de subtitel 'Deel 1'; deel twee gaat dieper in op de theoretische achtergronden van de Clipper-taal en op de technieken van objectgeoriënteerd programmeren. Deel één is duidelijk gericht op het ontwikkelen van grote applicaties in de praktijk; in hoofdstuk drie wordt dan ook een SYBEX-voorbeeldprogramma ontwikkeld om de administratie van deze uitgeverij bij te houden. De ontwikkeling van dit programma neemt inclusief enige tekst en uitleg vijftig pagina's in beslag. In de rest van het boek wordt voornamelijk ingegaan op de hulpmiddelen die bij de Clipper-compiler horen, iets wat eigenlijk ook in de vorige boeken is behandeld. Alleen hoofdstuk tien gaat nog over programma-onderhoud, maar dan alleen met behulp van RMAKE. Eigenlijk gaan alleen de eerste drie hoofdstukken over programma-ontwikkeling met Clipper, de rest is weinig verschillend van andere boeken en de Clipper-handleidingen. Het is dan toch jammer dat van een 'Werken met'-boek slechts zo'n 100 pagina's ofwel minder dan 25% gewijd is aan het ontwikkelen van projecten.

Titel: Werken met Clipper 5.0
Auteur: Martin Labee
Taal: Nederlands
Uitgeverij: Sybex BV, Soest
Omvang: 450 pagina's
ISBN: 90-5160-278-2
Prijs: *f* 89,--

Boeken over C++

Using C++

Bruce Eckel, de auteur van dit boek, is een ervaren C++ gebruiker en lid van het ANSI C++ comité. Het boek behandelt C++ versie 1.2 met behulp van de Glockenspiel en Zortech C++ compilers. Het boek is dus in feite gebaseerd op een oude C++ standaard, hoewel de eigenschappen van 2.0 kort aan bod komen. In het eerste hoofdstuk wordt objectgeoriënteerd programmeren behandeld, terwijl in hoofdstuk twee het gebruik van voorgedefinieerde classes wordt besproken. Veel aandacht wordt geschonken aan het creëeren van classes en de interne werking hiervan, wat in C++ erg belangrijk is. Complete voorbeelden worden gevonden in hoofdstuk 10, en in de bijlagen. Voor iedereen die verder wil in de C++ taal en die meer wil weten over de creatie van objecten in C++, is dit een zeer aanbevelenswaardig boek. Een sterk punt van het boek is dat ook gevorderde C++ programmeurs er iets aan hebben, zodat het boek ook na de beginnersfase waardevol blijft.

Titel: Using C++
Auteur: Bruce Eckel
Taal: Engels
Uitgeverij: Osborne-McGraw Hill, Berkeley, CA
Omvang: 617 pagina's
ISBN: 0-07-881522-3
Prijs: $24.95

Turbo C++ Programming

Reeds op pagina twee van dit boek wordt inheritance (overerving) in C++ uitgelegd, inclusief source-code voorbeelden. Pas daarna wordt ingegaan op de methoden, encapsulation enzovoort. Dit is op zijn zachtst gezegd een vreemde indeling, die de leesbaarheid niet bevordert. Ook de manier waarop source-code wordt gebruikt geeft aanleiding tot vraagtekens: er wordt geen gebruik gemaakt van aparte .cpp bestanden in een project, maar het hoofdprogramma neemt alle source-modulen op door middel van het #include directive. Tevens wordt het woord 'static' op sommige plaatsen in de tekst verkeerd gebruikt. De voorbeelden hebben in de meeste gevallen betrekking op grafische toepassingen zoals scrollbars en push-down buttons. Al met al is dit boek niet aan te raden als leerboek, maar ook gevorderde C++ programmeurs zullen het niet echt bruikbaar vinden.

Titel: Turbo C++ Programming
Auteur: Ben Ezzell
Taal: Engels
Uitgeverij: Addison-Wesley, Reading, MA
Omvang: 420 pagina's (diskette ingesloten)
ISBN: 0-201-55051-2
Prijs: $ 34.95

Mastering C++

Dit boekje, dat de subtitel 'An introduction to C++ and object-oriented programming for C and Pascal programmers' draagt, behandelt de meeste eigenschappen van C++ in circa 270 pagina's. In de eerste zeven hoofdstukken worden data, functies, pointers, classes en overloading besproken. Aparte hoofdstukken zijn gewijd aan in- en uitvoer met behulp van de stream-library en aan geheugenbeheer. Inheritance wordt in hoofdstuk tien besproken, terwijl de overige drie hoofdstukken gewijd zijn aan modulen, generieke classes en object-georiënteerd ontwerpen. Een aardige eigenschap van het boek is dat aparte 'notes' zijn aangebracht voor C- en Pascal-programmeurs, zodat de nieuwe eigenschappen vergeleken kunnen worden met de vertrouwde aanpak. Hoewel de aanpak van het boek in het algemeen vrij academisch is (de auteur is dan ook verbonden aan de San Jose State University), worden veel technieken duidelijk besproken, ook technieken die in andere boeken niet of nauwelijks aan de orde komen. Een klein nadeel van het boek is dat de voorbeelden weinig variatie verto-

nen; veel gelinkte lijsten, strings en matrices en dergelijke classes worden als voorbeeld gebruikt. Het boek is niet afhankelijk van een bepaalde C++ compiler, en is mede dankzij de opgaven ook geschikt als leerboek bij onderwijsinstellingen.

Titel: Mastering C++
Auteur: Cay Horstmann
Taal: Engels
Uitgeverij: Wiley & Sons, New York, NY
Omvang: 278 pagina's
ISBN: 0-471-52257-0
Prijs: ca. $ 32.00

Werken met C++

Een boek dat de taal C++ behandelt aan de hand van de Zortech C++ compiler, is Werken met C++. In het eerste hoofdstuk wordt een beknopt overzicht gegeven van de taal C, waarop C++ gebaseerd is. Object-georiënteerd programmeren en de eerste beginselen van C++ worden in hoofdstuk twee uitgewerkt, waarop in hoofdstuk drie wordt doorgegaan. In deze hoofdstukken worden classes, constructors, destructors, de algemene verbeteringen van C++ op C, friends en overloading besproken. Hoofdstuk vier is geheel gewijd aan de stream library, terwijl hoofdstuk vijf inheritance behandelt. Het boek is vrij praktijk-gericht, en bevat vele voorbeelden van mogelijke classes in de laatste vier hoofdstukken. Naast de bekende zaken als abstracte datastructuren wordt ook een dBase-class ontwikkeld, een interface naar de Clipper compiler en enkele rekenkundige classes. In hoofdstuk negen worden de library-functies van de Zortech compiler besproken, terwijl ook een muis-class wordt ontwikkeld. Het laatste hoofdstuk bevat een vrij uitgebreide class-library voor grafische toepassingen, waarmee tevens een voorbeeldprogramma wordt ontwikkeld. In de bijlagen wordt onder andere de 'name mangling'-techniek besproken. Deze wordt door C++ compilers gebruikt om identifiers mee te versleutelen naar unieke interne namen, om function en operator overloading mogelijk te maken. Dit boek is geschikt voor zowel beginnende als gevorderde C++ programmeurs die willen weten welke toepassingen er met C++ geschreven kunnen worden. Alle voorbeelden zijn verkrijgbaar op een aparte diskette.

Titel: Werken Met C++
Auteur: John Caspers
Uitgeverij: Sybex BV, Soest
Omvang: 388 pagina's
ISBN: 90-5160-174-3
Prijs: *f* 69,—

Programming in C++

Een vrij moeilijk te lezen boek met veel verschillende voorbeeldclasses, en net als 'Mastering C++' compiler-onafhankelijk en vooral gericht op studenten en professionele programmeurs. De voorbeelden variëren van vliegveld-simulaties tot gelinkte lijsten, en zijn gebaseerd op C++ versie 2.0. De indeling had iets duidelijker gekund, maar de inhoud is van hoge kwaliteit en behandelt de meeste eigenschappen en 'pitfalls' van C++. In tegenstelling tot Horstmanns boek is 'Programming in C++' echter minder geschikt voor de beginnende C++ programmeur.

Titel: Programming in C++
Auteurs: Stephen Dewhurst en Kathy Stark
Taal: Engels
Uitgeverij: Prentice Hall, Englewood Cliffs, NJ
Omvang: 233 pagina's
ISBN: 0-13-723156-3
Prijs: *f* 54,—

The Annotated C++ Reference Manual

Dit standaardwerk van de schepper van C++, Bjarne Stroustrup, is een must voor elke C++ expert. In de wandelgangen 'ARM' genoemd, biedt dit boek een uitweg voor de meest obscure problemen die tijdens het gebruik van de C++ taal kunnen optreden. Het boek is door het ANSI C++ comité gekozen als 'base document' voor de standaardisatie van de taal; tevens zijn voorgestelde uitbreidingen zoals templates (waarmee genericity mogelijk wordt) en exception handling vorig jaar door het comité goedgekeurd. Van vele technieken wordt uitgelegd waarop juist

voor die aanpak is gekozen, en niet voor de alternatieven. Dit en de grondige behandeling van alle elementen van C++ maakt de ARM vooral geschikt voor compiler-bouwers en professionele C++ programmeurs.

Titel: The Annotated C++ Reference Manual
Auteurs: Margaret Ellis en Bjarne Stroustrup
Taal: Engels
Uitgeverij: Addison Wesley, Reading, MA
Omvang: 447 pagina's
ISBN: 0-201-51459-1
Prijs: *f* 89,—

Turbo C++

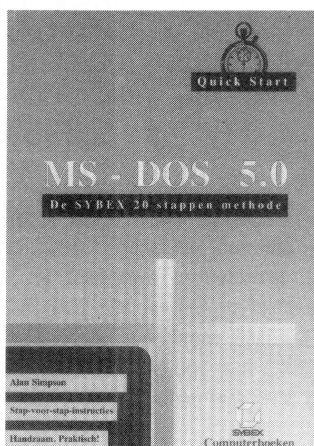
Specifiek voor de Turbo C++ compiler is dit boek van Pim Philipse, uitgebracht door Kluwer technische boeken. In het tweede hoofdstuk wordt onder andere de taal C++ puntsgewijs behandeld. Voor de gevorderde gebruiker een leuk overzicht, maar zeker te beknopt voor de beginner. Ook in hoofdstuk vier wordt de taal uitgelegd, maar weer erg beknopt. Het uitleggen van classes, waar het object-georiënteerde programmeren van C++ om draait, gebeurt bijvoorbeeld in slechts 7 pagina's! Dit is dus geen boek om C++ uit te leren, maar in feite een verkapte handleiding. In honderd pagina's worden de library-functies behandeld, iets wat in de geïntegreerde omgeving van Turbo C++ onder de functietoets Ctrl-F1 ook zeer uitgebreid aanwezig is.

In tegenstelling tot andere boeken van Philipse, beslaan de voorbeelden in dit boek slechts ongeveer 15 pagnia's; hierin wordt een string-class, een bestandsclass en een class om windows met streams te combineren ontwikkeld. Leuke voorbeelden die ook goed doordacht zijn, en jammer dat hier in het boek niet wat meer aandacht wordt besteed. Dit is eigenlijk het grote nadeel van het boek: Turbo C++ gebruikers hebben weinig behoefte aan een complete lijst van alle library-functies (die immers ook in de handboeken staan en online opvraagbaar zijn), maar des te meer behoefte aan een duidelijke behandeling van de taal C++ met veel verschillende voorbeelden. Voor de beginnende C++ programmeur is dit boek dus eigenlijk niet geschikt.

Titel: Turbo C++
Auteur: Pim Philipse
Uitg: Kluwer Technische Boeken, Deventer
Omvang: 303 pagina's
ISBN: 90-201-2502-8
Prijs: *f* 49,50

QuickStart MS-DOS 5.0

De voordelen en veranderingen van versie



5.0 zijn in een beknopte manier samengevat in deze QuickStart, zodat u na het installeren meteen aan de slag kan. Er wordt uitgegaan van een redelijke basiskennis van MS-DOS, maar er zullen weinig gebruikers zijn die de terminologie niet kunnen volgen. Een QuickStart van SYBEX bestaat uit twintig stappen, waarbij hier de eerste tien zijn gereserveerd voor het gebruik van de grafische shell van DOS 5.0. Erg belangrijk is de stap die handelt over hoe optimaal gebruik te maken van de nieuwe geheugentechnieken. Ook het terughalen van gewiste bestanden en het ongedaan maken van een formattering van een schijf komt aan de orde. Bent u nog niet in het bezit van DOS 5.0, en u wilt weten wat de nieuwe mogelijkheden zijn alvorens het te kopen, dan heeft u aan dit boek een goede handleiding.

Titel: QuickStart MS-DOS 5.0
Auteur: Alan Simpson
Uitgeverij: SYBEX Uitgeverij BV Soest
ISBN: 90-5160-299-5
Prijs: *f* 29,95

database

DB 3.0 PC-DBMS
Een compleet relationeel database management systeem. Het kan bestanden combineren, waardoor de nodige invoertijd en diskruimte aanmerkelijk bekort worden

DB 5.3/DB 6.3 File Express (2 disks)
Minimaal 384 Kb benodigd. Een krachtig, snel DBMS-systeem. Echter ontbreken hier de relationele rapporteringsmogelijkheden van PC-file+, waardoor het programma wel sneller te leren is voor de beginner of occasionele gebruiker. Een limiet (?) van 16.000.000 records, 120 velden per record, 250 tekens per veld. Veel mogelijkheden."5 sterrenschijf"

DB 7.3 Adreslabels
Een compleet mailing/postmanagementsyteem, voor het produceren van naam- en adreslabels. Snel en makkelijk in het gebruik. Adressen kunnen op allerlei manieren gesorteerd worden. Spoort dubbel ingevoerde records in bestanden op en dat kan veel werk besparen. Het heeft ook een intern mail merge systeem om standaardbrieven te maken.

DB 8.0 pBase DBMS
Programmeerbaar relationeel database management systeem. Het concept lijkt op dBase II, maar is wel meer gestructureerd, ongeveer zoals SQL, IBM's Query aanpak die zo in de belangstelling staat. Met onder meer onkosten-, inventaris- en verzendlijst-programma's.

DB 9.0 Instant recall
Een vrije vorm database management systeem als TSR (Ramresident) programma, dus altijd snel beschikbaar. Meerdere databases tegelijk open. Zeer sterke opzoektechnieken en schuiven met data en screens. ASCII import en export, maximaal 80KB files.

DB 10.0 PC-LIT
Een programma speciaal bestemd voor het indexeren van tijdschriftartikelen in een archiefkast. Het is in het algemeen geschikt voor het indexeren van genummerde artikelen. Dit is een DBMS-toepassing die gemakkelijk kan zijn voor bibliotheken, informatie- en knipseldiensten.

DB 11.3 Wampum
Een fraai menu-bestuurd systeem om dBase III te gebruiken. Hiermee kan zelfs de amateur schitterende toepassingen gaan ontwikkelen. Compleet met geavanceerde functies als audit trial, memo velden, edit checks, macro's en mail merge form letters. Dit programma is toegerust voor network support. Met wachtwoordbeveiliging. 512 KB en harde schijf vereist.

DB 18.0 Catalog reference
Dit systeem is speciaal ontworpen voor het catalogiseren van boeken, kranten, tapes, cassettes, diskettes, films etc. Krachtige zoekmogelijkheden. Geschreven in dBase III. Nodig is een harde schijf of twee floppy drives.

DB 25.0 Free File
Een buitengewoon gebruiksvriendelijk database-systeem wat de gebruiker binnen de kortse keren in staat stelt een database te creeeren entebeheren. Uitgebreide helpfuncties zijn ingebouwd.Draait op alle monitortypes. Max 2 miljard(!) records en 1000 velden per bestand. Min. 256 Kb en 1 (hard) diskdrive. Zeker aan te raden voor beginners !

●

DR 1 Painter's Apprentice

DR 29.0 Finger Paint
Tekenprogramma van grote klasse. Vergelijkbaar met Dr Halo, PC Paint etc. Geschikt voor CGA/EGA en Hercules. Muis optioneel. Voorzien van zes lettersfonts in 81 formaten. Een van de uitschieters uit onze Busware-reeks

PC BUSWARE

DR 30.0 Picture This
Tekenprogramma geschikt om tekeningen in POSTSCRIPT-formaat te ontwerpen. Voorzien van zoom- en lijndiktecontrole. Het voordeel van EPS-(Encapsulated PostScript) tekeningen is de afwezigheid van de gekartelde lijnen bij de tekeningen. De schijf bevat tevens Capture This, een CGA screen-capture-programma.

DT 2.0 dBAse II systems
Voor de aanhangers van dBase II een aantal volledig uitgewerkte toepassingen die ook zelf kunnen worden aangepast. Een time billing systeem, mailinglist programma, genealogy programma, rolodex simulator en Gourmet voor receptenbeheer.

DT 3.0 dBase III 1
Support tools voor dBase III van Ashton Tate. Onmisbaar voor de serieuze databaas. Met veel printerhulpjes, dBase in kleur, een ingekorte Wampum, flowfunctie, menumaker, etc. Ook met een paar uitgewerkte voorbeelden.

DT 4.0 dBase III 2
Een aantal goede utilities voor gebruik met dBase III, waardoor de mogelijkheden van een goed commercieel systeem sterk uitgebreid kunnen worden. De diskette bevat o.a. een koppeling naar programmeertaal C, een screen editor, en data dictionary DBSTRUPO. De screen editor is in feite een applicatie-generator die programma- en screenfiles aanmaakt.

onderwijs

EC 3.1 Typing Teachers (v/h ED 8.0) (CGA/ BASIC)
Bevat twee typecursussen. Behalve de normale typemachinetoetsen komen ook de specifieke pc- keyboardtoetsen aan bod. Bevat verschillende oefeningen op diverse niveau's, snelheidstests, typefout signaleringsroutines en accuratessebeoordelingen/

ED 4.0 PC Education 2
Natuurlijk moet je ook met de PC leren omgaan en wat is dan leuker dan dat met de PC zelf te leren. Dit is een complete interactieve les. Onderwerpen als toetsenbord, terminologie, I/O, DOS kommando's en het gebruik van de BAT-com. taal komen aan de orde. Helaas alleen voor wie goed thuis is in het Engels, maar dan ook grote klasse!

ED 5.0 Elementary Ed.1
Educatieve software voor de kinderen. De rekenspelletjes zijn natuurlijk universeel, voor de woorden is enige kennis van het Engels meegenomen. De computer als hulpleraar. Deze schijf hoort men zeker te hebben wanneer er kleine kinderen in huis zijn. Basic nodig.

ED 6.0 Teacher's Tools
Dit is een database systeem voor het bijhouden van de gegevens, cijfers en resultaten van de klas. Aanpassing van het VS systeem van cijfers geven is mogelijk. Dit is iets voor de leerkracht met een computer.

ED 14.0 Elementary Ed. 2.
Speelse leerprogramma's om kinderen in de leeftijd van 4 tot 9 met en door de computer te laten leren. Gezien de Engelstalige aanpak in ons land misschien iets voor oudere kinderen, de rekenspelletjes zijn meer universeel.(CGA)

EJ 5.0 Flags of the world
Versie 6.04 Een kleurrijke database van maar liefst 275 vlaggen van over de hele wereld; zowel landen als de bekendere organisaties (E.E.G. ect.) Tevens bevat deze schijf ook 50 volksliederen van meest bekende landen. Als U ze allemaal denkt te kennen: In het programma zit een herkenningsquiz ingebouwd.
Echt leuk programma. Helaas alleen EGA/VGA compatibel.

EN 11A.0 & EN 11B.0 NEC Database
Data van de Nat. Electric Code 1984. Data kunnen toegevoegd worden. Met o.a. 1- en 3-fase transformers. (Harddisk en 640 Kb ram vereist).

EN 12.0 Motorola Database
Motorola's complete discrete device offering. Met meer dan 7000 items, 58 catagorieen en 20.000 cross-references. Eenvoudig te gebruiken.

FO 3 A FF Forth 1

FO 3 B FF Forth 2

spel II

GA 1.0 Games 1
Verschillende spelletjes. (CGA/ EMULATOR) Schaken: spelkwaliteit redelijk goed, spelniveau is in te stellen tussen 1 en 6. Verder bevat de schijf een kruiswoordpuzzle, Vier op een Rij, het spel "PENTE" (een GOMOKU-achtig spel), Hi-Q en "LAB".

GA 8.2 Adventure system
Het spelen van een computerspel op de PC is leuk, zelf zoiets maken nog leuker en leerzamer. Dit pakket is voor het maken van tamelijk ingewikkelde Adventure Games, dus begin maar met het verzinnen van enge situaties en welke vragen daar gesteld kunnen worden. Er is natuurlijk ook een voorbeeld-spel bij, dus u heeft altijd wel iets aan deze schijf. Een beetje programmeerervaring is wel nodig.

GA 16.0 Classic Games
Games, geschikt voor CGA/EGA en Monochrome (met CGA_emulator) monitoren. Bevat o.a. Monopoly, Dammen, Backgammon en een paar kaartspelletjes.

GA 22.0 Bridge
Eindelijk een goed bridge-spel. Speel een hand, de computer speelt de andere drie. Het is ook mogelijk dit spel met twee spelers te spelen. De computer houdt de score bij. Veel conventies worden herkend.

GE 2.1 Family Ties
Dit mooie genealogy programma is goedgekeurd door de Latter Day Saints (Mormonen). Rapportage kan geschieden op de officiele LDS-manier of op de standaard manier, die minder formeel is. Dit systeem behandelt een moeilijk onderwerp maar is door de vriendelijke gebruikers-interface makkelijk te besturen.

GP 3.0 Pictures 3
Een uitgebreide compilatie van diverse plaatjes, in te laden in o.m. Timeworks Desktop Publisher, Ventura Publisher en Paintbrush. In totaal 297 stuks ! In .PCC formaat.

GP 4.0 Computer Logos
Een veelheid (108) aan gescande computeremblemen, in Dr Halo .CUT formaat. Tevens 17 lijntekeningen van computer-gerelateerde onderwerpen.

GP 6.0 Drop Cap
Zesentwintig letters in .Img formaat (lettertype Helvetica), in te laden in ondermeer Ventura Publisher en Timeworks Desktop Publisher. Naast deze letters bevat de schijf een assortiment pijlen en andere "richtingaanduiders" in .Img formaat.

GP 7.0 Symbols
78 Algemene plaatjes en symbolen met betrekking op o.a. astrologie, medicijnen, winkels en reizen. Gescand in Dr Halo .cut formaat en geschikt voor o.a. Ventura en Timeworks.

GP 8.0 Old English Capitals
26 Hoofdletters zowel in PC Paintprush formaat (.PCC) en GEM Image formaat (.IMG)

GP 10.0 Whiplash Clipart
19 verschillende tekeningenbestanden in MacPaint formaat (.MAC) Deze plaatjes onderscheiden zich door hun buitengewoon goede scherpte. Eventueel overzetbaar im Ventura / Timeworks .IMG formaat.

grafisch

GR 7.0 Picture viewers
Een compilatie van verschillende programma's welke de diverse courante (en minder courante) plaatjesformaten op het beeldscherm toont. Om maar eens wat formaten te noemen: *.GIF ; Mackintosh *.RLE voor Hercules t/m VGA. Verder PCX, CUT , PIC IFF (deLuxePaint) en SCX Monitorcompatibliteit afhankelijk van het gebruikte programma (7 stuks)

GR 8.0 Draftsman
Draftsman is een geavanceerd zakelijk grafisch systeem, dus voor "Business Graphics", de bekende bar, pie (taart), exploded pie, line etc. Alle grafiekvormen zijn mogelijk. Titels en legenda's kunnen erin verwerkt worden, lijnen getrokken etc. DIF file import is mogelijk, dus ook vanuit Lotus en dBase interfacing. Zeer goed programma van professioneel niveau. Uitdraai via diapresentatie op Epson printer of HP 7470 plotters. CGA nodig

GR 9.1Presentation Graphics
Present is een prima programma voor het maken van diashow-presentaties, waarbij het mogelijk is om zowel tekst als grafische informatie uit andere programma's toe te passen. Het is mogelijk met zowel een kleuren-, als met een monochrome monitor te werken. Het pakket kan 12 verschillende soorten fade-ins maken en kan de samengestelde presentatie zowel automatisch als handmatig weergeven. Er is een screencapture bij en de files zijn compatible met Basic Bsave files.

GR 10.0 Dancad 3D
Een uitgebreid driedimensionaal (wire frame) tekenen grafisch programma. Het werkt op een standaard graphics monitor. Het hoofdprogramma geeft een kijk vanaf de voorkant, bovenkant en vanuit verschillende perspectieven. Printen, met hoge resolutie, kan op Epson printers of verschillende plotters. Er is 512 KB nodig, CGA en enige programmeerervaring.

GR 11.0 Slide Shows
Een erg goed programma op deze schijf is GRASP, een presentatiesysteem met screenplay utility en slide show funkties.Compatible met PC Paint en BSA VE picture formaten. Er staan ook wat achtergrondfoto's/beelden op, die voorzien kunnen worden van tekst. Met onder andere de bekende plaatjes van space shuttles, vliegtuigen en computers. Nodig is een CGA, EGA of Hercules kaart.

GR 12.0 Turbo Paint
Dit programma is vergelijkbaar met de commercile tekenprogramma's die er op de markt zijn. Nodig is een 256 K CGA kaart en een muis die compatible is met Microsoft of Mouse Systems.

GR 13.1 Molecular Modelling (CGA/EGA)

Heel geschikt als educatief programma voor scheikunde. Zelf via de computer (roterende) moleculen samenstellen. Voorbeelden zijn al aanwezig, ieder atoom heeft een eigen kleur. Ingebouwd periodiek systeem. Daarnaast nog een soortgelijk programma voor de EGA monitor.

GR 16.0 PC-Draft I

Een programma dat vergelijkbaar is met PC Paint, PC Paintbrush etc. Alle normale opties plus de mogelijkheid om via een speciale routine ingevoerde gegevens grafisch te manipuleren. De resolutie is 240 dpi op Epson printers en 150 of 300 dpi op Laserjets. Vereist is CGA, 512K. Microsoft muis wordt aanbevolen.

GR 26.0 Optiks

Tekenprogramma / grafische spreadsheet. Compatible met EGA/CGA/VGA en Hercules, en plusminus 30 grafische formaten w.o. .CUT, .PIC, .PCC, PCX, en .MAC. Printeroutput naar de meeste bekende printers. Te besturen met muis en/of Keyboard. 80 kilobyte manual op disk. Kan ook ASCII-bestanden laden.

●

GS 1.0 Simulations

Zes simulatiespellen. O.a.: Airtrax- beman de radar van een verkeerstoren, en wees een zo goed mogelijke verkeersleider. Net als in het echt geen salarisverhoging mogelijk! Bigrig- Cross-country trucking. Niet vergeten te slapen en/of tanken. Fire-Blus de bosbrand. US- Een wargame. Oilwell- Boren naar olie, raffineren en verkopen.

GT 2.0 Sorta Star Trek

Een goede DRIEDIMENSIONALE versie van het klassieke spel. (CGA) Te spelen in een kubus (= 3-D) i.p.v. een plat vlak (= 2-D). Tevens het Startrek Trivia spel, gebaseerd op de bioscoopfilms, -series en nieuws van fanclubs.
GV 12.0 Adventure Solutions
Hints en tekstbestanden met op lossingen voor vele adventures o.a. King's Quest, Colossal Cave en Questron. Betreft in totaal meer dan dertig adventures.

HA 2.0 Jokes II

Deze schijf bevat twee "opmerkelijke" DOS-versies (LOVE-DOS en RUDE-DOS) welke erg geschikt zijn om kennissen en/of collega's te verrassen. Naast deze nog een nep MacIntosh-emulator en nog wat memory-residente monsters, letterhappertjes en meer plagerijtjes. *Niet te gebruiken bij mensen met hartklachten.*

HA 3.0 Indiana Jokes

Zie beschrijving in catalogus onder HA 1.0

HK 2.0 Wine Cellar

Database, speciaal ontworpen voor de wijnliefhebber / wijnkelder.
Registreert op oorsprong, kleur, jaar, botteldatum, aankoopdatum, prijs, alcoholpercentage en commentaar. Zoek- en printroutines ingebouwd. 220 wijnen reeds ingevoerd. Draait op vrijwel ieder monitortype.

HP 1.1 Health (BASIC)

Bevat o.a. Bioritme programma, Glycemy, een programma wat bloedsuikerspiegels graphisch weergeeft op de plotter. Verder Lifegame, een programma wat aan de hand van Uw leefstijl en gewoontes berekent wat Uw te verwachten levensduur (!) zou kunnen zijn. Als laatste de twee programma's SUBLIM EN TPNCALC, vrij specialistisch-medische programma's om proteine/elektrolyten factors te berekenen.

HP 9.0 Astrologie (Hercules/CGA)

Behoorlijk uitgebreid astrologieprogramma. Berekent o.a. "Radixhoroscopen volgens het Koch- of Placidussysteem" Verder: aspecten, transits, synastrie, Weergave van de radix op het scherm en op de (Epson/IBM) printer Bevat een uitgebreide handleiding. Zonder (geringe) kennis van de astrologie niet echt aan te raden. Dan liever HP 18.0

PC BUSWARE

HP 10.0 Movie Database

Bevat reeds meer dan vierduizend films met o.a. acteur, schrijver, oscars, regisseur, producer, jaar van productie. De database is zelf nog verder uit te breiden met andere films en gegevens. De database is uitgerust met 'n speciaal zoekprogramma.

HP 12.0 Keukenhulpen

Een receptensysteem met maaltijdplanning en boodschappenlijsten. Verder is er Mealmate, dat de calorieën en voedingswaarden van 800 (Amerikaanse) voedselprodukten, waaronder 21 soorten kaas, weergeeft.

HP 16.0 VCR Database

Administratieprogramma voor videocassettes met o.a. cassettenummer, datum, volgorde, bandteller, titel, categorie, namen enz. Bevat tevens een labelprintroutine.

HP 18.0 Personalities

Drie programma's: -Het handschriftanalyseprogramma doet na 13 vragen m.b.t. een handschrift de persoonlijkheid van de schrijver uit de doeken. "Redelijk accuraat" volgens ons proefkonijn. -Het Bioritmeprogramma: voor diegenen die vinden dat dit programma niet thuishoort op schijf HP 1.1 .-Het Numerologieprogramma: volgens een getallenleer rekent dit programma gegevens zoals geboortedata en naam om tot een uniek getal aan de hand waarvan diverse berekeningen m.b.t. toekomstvoorspelling, karakteranalyse, temperament enz. mogelijk zijn.

HP 35.0 Oracle

I Tjing programma, gebaseerd op de oude chinese techniek voor wat betreft het voorspellen van de toekomst of het raad geven op vragen. Naar keuze te werken met de munten of stokjes. De verkregen resultaten kunnen naar keus naar scherm of printer gestuurd worden. CGA/EGA/Hercules.
Tarot. Een minstens zo bekend systeem dat met de kaarten werkt. De uitkomsten van dit orakel kunnen naar printer, scherm of disk.
CGA/Hercules.

IN 7.0 Aandelenbeheer 2

Voor het in de gaten houden van de aandelenkoersen, het beoordelen van de eigen posities en het nemen van koop- en verkoopbeslissingen. Met kaartsysteem en goede grafiekweergave. Met download programma's voor Amerikaanse databanken zoals Dow Jones, die eventueel via Memocom en de Source ook van Nederland uit aangeroepen kunnen worden. CGA kaart nodig.

HS 22 a/b/c Marine Navigation (3 disks)

Een aantal utilities specifiek voor schippers en anderszins varenden. O.a. berekeningen voor tijdstip van zonsondergang en zonsopgang, koers, afstand, rendez-vouz-snelheid, tijdzone- conversie, interceptiekoers.

LB 8.0 Baker's Dozen

Een menugestuurd utilitysysteem met de functies: Mini- Spreadsheet, kalender (met feestdagen van diverse landen en religie's), bestanden zoeken, textbestand vergelijking, screencapture, com 1 en com 2 wisselaar, FAT uitlezer/editor, gekanteld printen van text. Mag zeker niet ontbreken in de Utilities-Directory!

LB 9.0 SMG Utility Set

Verzameling van 30 (!) utilities, alle vergezeld van .DOC-betanden waarin tekst en uitleg. O.a. DOS-Plus, Fix-Disk, Hard-Disk info ListBad (traceerd kapotte clusters), Owner (traceert clusters bij files), ListDir, RememDir en SaveDir. Buiten deze nog meer, teveel om op te noemen. Zeker aan te bevelen voor de actieve hard-disk gebruiker.

LO 1.1Lotus Worksheets 1

Een twaalftal toepassingen voor gebruik met Lotus 1-2-3, waaronder het toevoegen van histogrammen, bioritmes, aandelenbeheer, leningen, trend-analyse, loonadministratie (tot 12 werknemers), verhuursysteem, etc.

LO 2.1 Lotus Utilities

Om gemakkelijker toegang tot Lotus te krijgen. Met drivers, communicatie, mini spreadsheet auditor en EMS emulatie.

LO 3.0 Lotus Worksheets 2.

Deze schijf bevat een conversie van Lotus formaat naar ASCII, een utility voor printer-control, een template voor gebruik met prokey en een heel goede tutorial voor 1-2-3 macro's.

LO 6.0 Lotus Worksheets 3

Een reeks uitgewerkte 1-2-3 work-sheets voor toepassingen als boekhouding, kasgeldbeheer, statistische bewerkingen, ordergrootte berekenen etc.

LO 9.0 Print Graph Library Lotus.

Dit is alleen voor de oude 1-2-3 versies (1 en 1a), waarmee meer printers, zoals laserprinters, HP, Star, IBM plotter, Canon inkjet worden toegevoegd.

LO 10.0 Goalseeker

Razendsnelle Lotus utility, voor het berekenen van cel-waarden, wanneer de eindwaarden bekend zijn.

MA 40.0 Mandelbrot

Fractaalgenerator. Maakt de wonderlijkste tekeningen op uw beeldscherm. Maakt gebruik van een coprocessor indien aanwezig.
Een handleiding op disk betreffende de wiskundige achtergronden van dit fenomeen verteld u meer. Werkt op Hercules/CGA/EGA/VGA.

talen

ML 4.1 XLISP

De LISP interpreter voor wie echt iets wil doen met deze AI taal. Compleet met source code in C, dus helemaal aan te passen. Er zijn voldoende voorbeelden toegevoegd, waaronder een interactieve Logo simulatie en een mini-Prolog interpreter, geschreven in XLISP.

ML 11.0 PC Demo System

Een compleet nieuw systeem voor het vervaardigen van demo's van bijvoorbeeld nieuwe softwareprogramma's. Het is mogelijk om tekst op schermen te tonen, in een diapresentatie of interactief lesprogramma. De presentatie kan door de gebruiker worden bestuurd of in een bepaalde volgorde worden getoond. Kleur en ANSI-graphics worden ondersteund. Het is in feite hetzelfde systeem als het Dan Bricklin Demosystem.

De juiste manier om de PC Busware diskettes snel geleverd te krijgen is het overmaken van het betreffende bedrag (*f* 7,50 of Bfr 200 per disk/ bestelt u meer diskettes tegelijkertijd dan bedraagt de prijs per 5.25" diskette *f* 5,00/ Bfr. 150/ prijs 3.5" *f*8,50/Bfr 250/ bij meerdere disks *f*6,-/Bfr 175) op gironummer 3157656 t.n.v. Infolist Amsterdam, Postbus 43048, 1009 ZA Amsterdam (voor België BBL310-0506025-62 t.n.v. SAC), met vermelding van het zendadres en de codenummers van de gewenste diskettes/formaat (5.25" of 3.5")

ML 13.0 Prolog

Deze Artificial Intelligence programmeertaal is niet gemakkelijk, maar een toekomstzekere tijdspassering. Dit is de versie van AD en biedt een vrij uitgebreid en kompleet Prolog systeem. De meegeleverde disk-dokumentatie is echter niet bedoeld als Prolog tutorial, maar om echte Prolog toepassingen te draaien.

ML 14.1 CP/M Emulator

Een 8088 CP/M nabootsing van Z80 CP/M 2.2 inkl. terminal VT 52 emulatie op de PC. Voor wie wel eens zijn PC wil gebruiken voor software uit een heel andere wereld. Niet alle diskdrives zijn echter geschikt voor het lezen van de PC/M formaten, slechts een deel kan gelezen worden via de normale schijfjes.

ML 17.0/ML 18.0 Expert System (2 disks)

Dit zijn twee schijven met een demonstratie programma voor de bouw van een expert-systeem. Het is een demonstratieversie want het werkt wel echt, alleen is het aantal "rules" dat men kan maken veel beperkter dan bij de complete versie.

PA 2.1 Turbo Pascal 1

Utilities en routines voor Turbo Pascal. Goed gedocumenteerd; bij de meeste utilities zijn voorbeelden opgenomen. Met o.a. picture string, datamanipulatie, grafische routines.

PA 3.3 Pascal Math

De wiskundige trucjes van Pascal kunnen hiermee eenvoudiger benut worden. Werkt met de 8087 mathematische coprocessor, zowel voor Turbo als Microsoft Pascal. Boordevol met statistische, wiskundige en technische routines en methoden. Iets voor technische studenten, maar ook voor de technisch ontwerper en ingenieur.

PA 9.1 Pascal Tutor (leermeester)

Pascal-instructieschijf, adequaat tot versie 4.0 van Borland's Turbo Pascal. In totaal bevat deze schijf 108 voorbeeldprogramma's in Turbo Pascal geschreven. Onderwerpen als loops, controls, procedures, arrays en type komen uitgebreid aan de orde. De instructie bestanden (380 Kb !) zijn uitprintbaar.

●

PA 11.0 Altamira

Tekenprogramma dat helemaal met het toetsenbord werkt en waarmee tekeningen gemaakt kunnen worden die direkt naar de printer gaan of worden gebruikt in Turbo Pascal programma's.

PA 16.1 Screen Engine

Uitstekend interactief ontwerpsysteem voor het ontwikkelen van data invoer, Lotus-achtig menu en hulpschermen voor Turbo Pascal. Met voorbeelden. Geschikt voor monochrome- en kleurenschermen. Zowel simpele als complexe boxen zijn makkelijk te tekenen en te veranderen terwijl u het scherm ontwerpt.

RD 16a t/m RD 16d Wisdom of the ages

Vier schijven gevuld met zegswijzen, uitspraken, epigrammen, en wijze woorden door de eeuwen heen. Ze bevatten in totaal rond de 6.500 uitspraken en deze kunnen op onderwerp geselecteerd worden. Onderverdeeld in drie secties: klassiek, middeleeuws en modern. Deze secties zijn afzonderlijk uit te schakelen, dan wel te combineren. Harddisk noodzakelijk. Hercules/CGA/EGA/VGA

SP 5.1 / 6.1 Express Calc (2 disks)

Een niet al te ingewikkeld Spreadsheetprogramma met maximaal 64 kolommen en 256 regels te gebruiken. Handig voor diegenen, die eens kennis willen maken met spreadsheets. Vergezeld van een buitengewoon uitgebreide handleiding van 190 pag.'s.

TE 5.0 NY edit

New York edit is een nieuwe tekst editor met tal van mogelijkheden. Vind- en vervang operaties, kolomen blok operaties, keyboard macro's, etc. NY edit kan gebruikt worden met twaalf open vensters tegelijk, waarbij elk venster een ander bestand bevat. Harde schijf is vereist.



Mindware

Uw computer als interactief medium om te communiceren met uw onderbewustzijn. Het klinkt in eerste instantie misschien raar, maar in de V.S. bestaat deze software al jaren en de gebruikers worden steeds enthousiaster. Uw computer luistert naar u, geeft advies en wijze raad, helpt u met het bedenken van allerlij nieuwe en creatieve ideeën en kan zelfs uw liefdes relatie analiseren. De software is alleen in voorraad op MS.Dos 5.25"

I.Q. Smarts — *f* 99,-

Voor het meten en het vergroten van uw I.Q. Print overzichten en rapporten uit en ontwikkelt trainingssessies aangepast aan uw behoeften.

Neurobics — *f* 139,-

Aerobics voor de geest! Een compilatie van oefeningen en puzzles, gepresenteerd in interessante graphics die uw hersenen een goede "workout geven". Een zeer geliefd programma.

The Idea Generator Plus — *f* 595,-

Dit programma haalt de betere ideeën uit de verste hoeken van het brein. Geprezen in de N.Y. Times. Gratis wordt bijgeleverd het boek 'The art of creative thinking'; een uitgebreide verhandeling over het proces van creatief denken.

Idea Fischer — *f* 1495,-

I.F. versterkt de kracht van uw ideeën. Het is het meest uitgebreide programma in z'n soort en wordt veel gebruikt door reclameburos, managers en de directies van grote bedrijven. Een uitgebreide demodisk kost *f* 75,-.

Speciale aanbieding
De PC Synergizer

Deze hardware/software combinatie maakt van uw PC een zeer krachtige brainmachine die men zowel voor privé als voor professionele / therapeutische doeleinden kan gebruiken bij ontspanning, stress- en slapeloosheidsbestrijding, meditatie, hypnose, concentratie- en creativiteitsverhoging, etc. De set bestaat uit een insteekkaart, software, een zéér mooie bril met 8 LED's en een uitgebreide manual. Met de PC Synergizer kunnen sessies geprogrammeerd worden van willekeurige lengte en complexiteit. Elk oor / oog kan apart gestimuleerd worden, waarbij licht- en geluidsfrequenties op elkaar aangepast zijn.
Nu voor de speciale prijs van *f* 995,- incl. BTW (normaal *f* 1395,-)

Calmpute — *f* 349,-

Uw computer als bio-feedback apparaat. U kunt de biologische informatie van uw huid lezen en interpreteren. Hiermee kunt u methoden van zelfcontrole (zoals stress management en ontspanning) leren en ontwikkelen.

Mind Mirror — *f* 99,-

Gedeeltelijk een stuk gereedschap gedeeltelijk filosoof op disk. De bekende Timothy Leary is computer evangelist geworden en zijn kennis en ervaring op het gebied van psychologie, bewustzijn etc vindt u op deze diskette terug.

Synchronicity — *f* 199,-

Intuïtie is vandaag de dag erkend als belangrijk element in een groot aantal succes-stories. Carl Jung ontdekte het geheim van het maken van de juiste beslissing op basis van intuïtie. Hij noemde dit synchronicity; het principe van het toeval met een bedoeling. Dit programma neemt u mee op reis in die wereld.

The Emotional Spreadsheet — *f* 495,-

Met dit programma ontdekt u een heel nieuwe benadering om uw lotsbestemming meer in eigen handen te nemen. Het laat u zien hoe ogenschijnlijk niets met elkaar te maken hebbende zaken en gebeurtenissen in uw leven toch een zekere relatie hebben. Het quantificeert de belangrijke gebeurtenissen in uw leven in een soort spreadsheet, die de steeds terugkerende patronen blootlegt.

Compatibility Prober — *f* 175,-

Hiermee kunt u op wetenschappelijke wijze de compatibiliteit van twee (of meer) mensen meten. Maak een relatieschema van u en uw baas, uw relatie, compagnon etc. Een van de betere uitgaven van 'Personality Software'.

Heart to Heart — *f* 149,-

Een relatie-analyse programma op het gebied van de liefde. Beide partners krijgen een vragenlijst (doelstellingen, sex, vrije tijd financiën etc) en via de database worden overeenkomsten en verschillen duidelijk.

PC Guru — *f* 349,-

Een programma waarmee u kunt converseren. Sex, schoonmoeders, relatieproblemen, woede; PC Guru staat overal voor open. Vocabulaire: 30.000 woorden en 5.000 idiomatische expressies.

Mentor — *f* 149,-

Naast oefeningen in probleem oplossen bevat Mentor routines om uw geheugen, mentale coördinatie, reactievermogen, creativiteit en perceptie te stimuleren / verbeteren. Mentor is ontwikkeld vanuit 2 achtergrondgedachten:
1. Het stelt de gebruiker in staat een evaluatie te maken in prive omgeving zonder de kosten van een beroepsmatige onderzoeker.
2. U kunt de tests en oefeningen herhalen wanneer u dat uitkomt.

Best Choise 3 — *f* 349,-

Wie moet ik in dienst nemen? Welke auto zal ik kopen? Wat voor software is het beste voor deze taak? Welke opdracht zal ik het eerst uitvoeren? Best Choise geeft suggesties opgebouwd uit 3 elementen: keuze, criteria en mensen.

Houdini — *f* 299,-

Ook een ideeën generator welke uit los ingevoerde stukjes informatie een overzichtelijk, aan elkaar te koppelen idee/informatie systeem kan brouwen. Zit in dezelfde sfeer als Idea Fischer maar is aanzienlijk goedkoper.

Wisdom of the Ages — *f* 275,-

Leen uw volgende punchline van de grootste geesten van onze aarde. 1000 filosofen, dichters, denkers, wiskundigen etc geven in 81 categoriën hun wereldberoemde uitspraken. Zeer geschikt voor het schrijven van speeches, opstellen, verhalen, folders etc.

Vraag de Catalogus aan!
Showroom: Nwe Kerkstraat 67 Amsterdam
Postadres: PB 75570 - 1070 AE Amsterdam
tel.: 020 - 6203219 / 6268069 fax: 020 - 6253280

PC ALMANAK



PC ALMANAK 1991: meer dan 300 pagina's actuele overzichten van het marktaanbod aan PC's (XT's, 386 AT's, 486 AT's), laptops, printers, monitoren, scanners, modems, software, dealers, opleidingen en meer.

PC ALMANAK 1991: alle prijzen plus uitgebreide technische specificaties overzichtelijk op een rij.

PC ALMANAK 1991: onmisbare informatie voor wie wil kunnen vergelijken alvorens aan te schaffen.

De PC Almanak ligt nu in de winkel
U kunt hem echter ook direct bestellen:
Maak ƒ 29,95 (incl. verzendkosten) over op rekeningnummer 1585491 t.n.v. Sala Communications Amsterdam o.v.v. uw naam, adres en "PC Almanak".